Chapitre01. Introduction to software Engineering

Software:

Software is a collection of instructions, data or programs that enable computers to perform specific tasks, acting as a bridge between hardware and users.

Components of software:

1) Project management Documents:

These documents help in the organization, planning and management of software project

They include: plans, budgets timelines, milstones .. etc

2) Specification:

This describes the list of functions that the software should fulfill.

It also includes quality attributes (exemple: scalability, robustness ... etc), constraints and environmental interfaces

3) Design:

Breaks the specifications into modules or objects, detailing their structure, interfaces, and algorithms.

4) Source code and Executable:

- Source code: Human-readable instructions written by programmers. It the code that get compiled to run the software
- Executable: it the compiled code that can be run directly on a computer by users.

Features of software:

1) Unique product: created once but can be replicated

2) Durable: issues arise from design flaws and not from wear and tear

3) Complex: addresses complex human problems

4) Invisible: creating software is purely a mental task, making it hard to judge its quality.

5) Evolving Techniques: continues to rely on handcrafted processes despite advancements.

Software Engineering:

It is the displine that involves the systematic design, development, testing and maintenance of software applications ans systems

It applies engineering principles to create a reliable, efficient and high-quality software solution.

Software Engineer:

A professional who applies engineering principles throughout the software developement process from design to maintenance.

Goals of software Engineering:

- Develop high-quality, cost-effective software.
- Use systematic processes for software creation and maintenance.
- Solve problems efficiently with good skills.
- Communicate effectively with customers to meet their requirements.
- Plan carefully, setting out what's needed, when and what to achieve
- Use tools and innovative methods for software developement.
- Continuously improve speed and quality in developement
- Apply engineering principles throught the software lifecycle.

Software Developement Lifecycle:

1) Need (requirement) analysis
2) specifications development
3) Modeling
4) Developement.
5) Testing
6) Maintenance

Principals of Software Engineering:

1) Separation of Concerns:
Organize software into subsections, each with a unique and well-defined responsibility.

2) Reuse:
Encourage the use of pre-built software components such a libraries, to reduce complexity, meet tight deadlines and maintains quality

3) Maximum Encapsulation:
Expose only what is strictly necessary for a subsection or class to fulfill its role.

4) Loose Coupling:
Minimize dependencies between components to enhance flexibility and reduce the impact of changes

5) Strong Cohesion:
Groupe together elements that perform similar roles or address the some issue, while avoiding the mixing of unrelated functions.

Characteristics of a good software:

1) Functionality: performs specified tasks effectively
2) Usability: User-friendly with intuitive interfaces
3) Efficiency: Optimizes resource usage.
4) Reliability: Consistent performance without crashes.

5) Maintainability: Easy to update and fix

6) Portability: Runs across different platforms

7) Security: protect against threats and unauthorized access.

8) Robustness: resilient to unexpected conditions.

9) Scalability: supports growth and high demand

Software Life Cycle:

Requirement Gathering and Understanding:

- Activities: Meetings with clients, clarifying requirements defining the product's purpose and users
- Output: SRS (Software Requirement Specification) documents.

Designe:

- Activities: Review SRS, decide on software architecture and system design.
- Output: software Design Document outlining the structure and operation of the software.

Coding:

- Activities: Write and optimize code based on the design document
- Output: working software components or modules

Testing:

- Activities: Test the software against SRS, identify and fix defects, perform retesting and regression testing
- Output: Well-tested software ready for deployement

Deployement:

- Activities: prepare the production environment, conduct User Acceptance Testing (UAT) and secure customer approval
- Output: Live software accessible to users

Maintenance:

- Activities: Monitor performance, resolve issues and implement enhancements
- Output: Updated software versions ensuring smooth functionality.

Software Lifecycle Models:

Waterfall Model:

- Process: Sequential, phase-by-phase approach
- Pros: Simple, clear structure
- Cons: Time-consuming, rigid for evolving requirements.

V-shaped Model:

- Process: Development and testing run in parallel
- Pros: High quality, good for fixed-scope projects
- Cons: Expensive, inflexible to change

Iterative Incremental Model:

- Process: Develops in iterations with continuous feedback
- Pros: Flexible, allows refinements
- Cons: Requires regular reviews.

Prototype Model:

- Process: Build and refine a prototype before final development
- Pros: Reduces costs, ensures customer clarity
- Cons: Scope changes can increase complexity.

Chapitre 02 : Modeling with UML

What is a Model :

It is a simplified representation of a real-world phénomenon, process or a system.

It is used to describe, explain or predict behavior while omitting less significant details to focus on essential aspects

Key Features of models :

- Simplify reality to help understanding.
- Helps align stakeholders on a solution.
- Translate real world problems into a longuage a computer can undestand

Types of models :

- Predictive Models : Forecast outcomes. Exemples :
  - Meteorological Model that helps predict the weather
  - Economic Model that helps predict the stock market
- Conceptual Models : Provides a structured overview.

Exemples :

- Plans : models that gives an overview of the concerned systeme

Modeling :

Modeling involves creating a simplified representation of a system to understand its operation, manage complexity and ensure coherence

Benefits of Modeling:

1) Simplifies complexity:
provides a manageable version of reality

2) Enhances Communication:
ensures mutual understanding between stakeholders and developers.

3) Improves efficiency:
supports automation (code generation) and maintenance.

4) Maintain quality:
Crucial for high-quality software and efficient updates post-developement

5) Supports iterative Develop-ment:
Eases transitions from conceptual design to coding

Analysis and design methodes:

Composition vs Decomposition:

- Ascending methods: Build system from existing models.
- Descending methods: decomp-ose the system into simpler programmable modules.

Functional vs Object-Oriented:

- Functional (process driven):
The system is viewed as hiero-rchical units with shared states
- Object-Oriented:
The system is viewed as inter-acting objects with decentral-ized states.

Purpose of Modeling:

Modeling ensures thes system's structure, operation and boundries are well-underst-ood, aiding in developing a high-quality software while reducing costs and delays

Object Oriented Modeling:

It is an approach that views software as a collection of separate objects with specific properties. These objects interact to perform the software functionalities

OOM focuses on the system's objects, not just what it should do

Key features of an object:

1) Identity: it maked the object unique
2) Attributes: Data describing the object's state
3) Methods: Functions or actions the object can perform, often linked to its attributes.

UML (Unified Modeling Language):

It is a visual, object-oriented modeling Longuge used to represent systems as objects or object-related concepts.

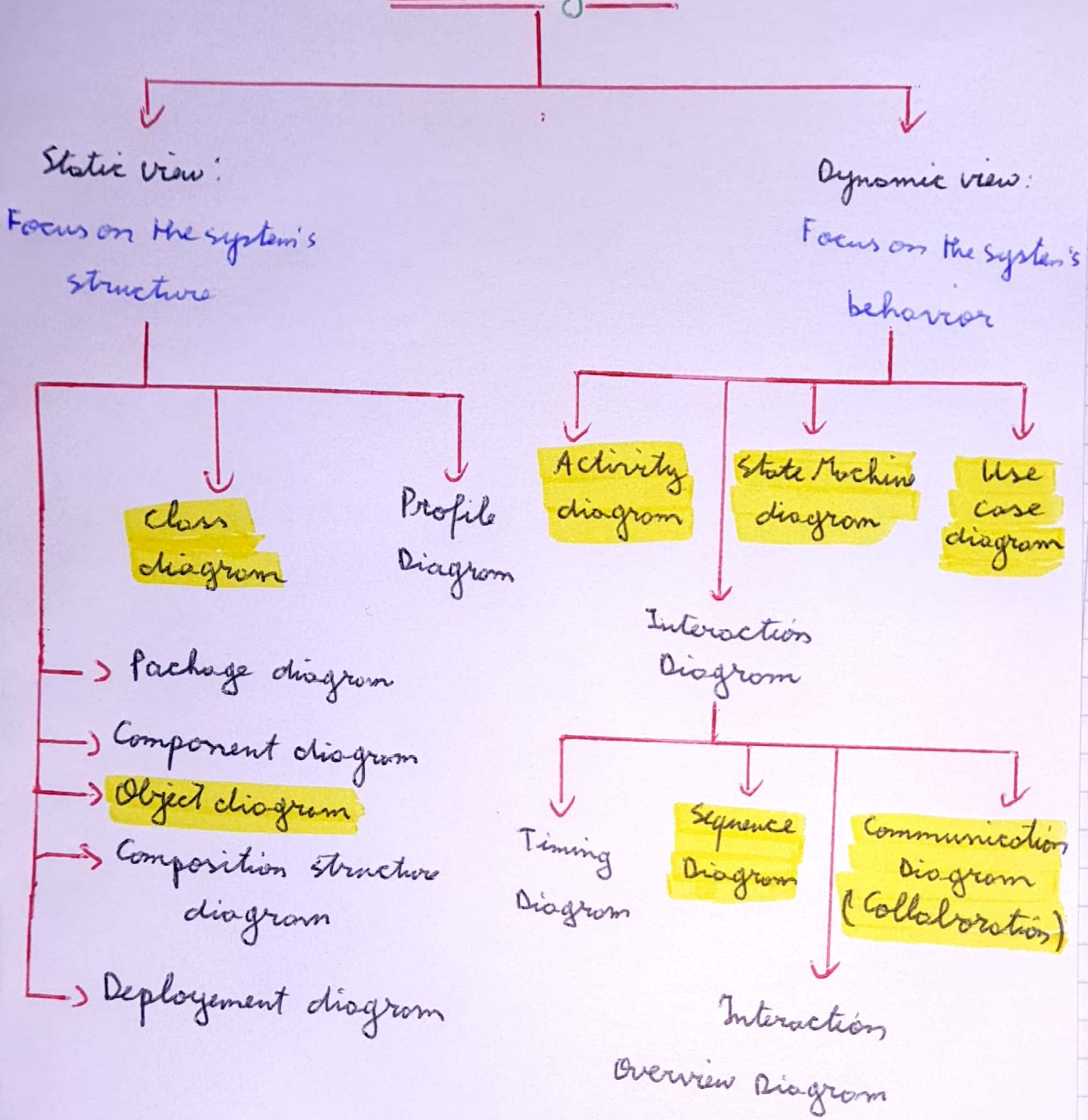
UML is not a method but a standard language widely adopted in the industry

It provides views and diagrams to visually design and understand systems.

General concepts:

- Class: An abstract data type defining shared properties for a groupe of objects.
- Encapsulation: Hides an object's internal details, exposing only an interface, simplifying updates.
- Inheritance: subclasses inherit properties from a main class
- Specialization / Generalization helps create class hierarchies
- Polymorphism: Enables methods to operate on objects from different classes, enhancing code flexibility.

UML Diagram


Static view:
Focus on the system's structure
Dynamic view:
Focus on the system's behavior
Class diagram
Profile Diagram
Package diagram
Component diagram
Object diagram
Composition structure diagram
Deployement diagram
Activity diagram
State Machine diagram
Use case diagram
Interaction Diagram
Timing Diagram
Sequence Diagram
Communication Diagram (Collaboration)
Interaction Overview Diagram