Models

1. **Waterfall Model**: The waterfall model is a linear approach to the software development life cycle. In this model, the phases of software development, such as requirements gathering, system design, implementation, testing, deployment, and maintenance, occur sequentially in a rigid order. Once a phase is completed, it cannot be revisited until the next phase has been started.

2. **Spiral Model:** The spiral model is an iterative software development process that combines elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. It is a risk-driven model that provides flexibility to accommodate changes throughout the development lifecycle while also ensuring that all aspects of software development are addressed.

3. **Unified Process Model (UPM):** UPM is an iterative software development process framework that provides a set of best practices, activities, and artifacts for organizing and managing software development projects. UPM emphasizes four key disciplines: business modelling, requirement management, analysis and design, and implementation. Each discipline contains several processes, which can be tailored to meet specific project needs

4. The **iterative-and-incremental model** is an adaptive approach where development is broken down into smaller, more manageable pieces. The process consists of repeating cycles of planning, design, implementation, testing, and evaluation, with each cycle building on the previous one. This approach is more flexible and can accommodate changes in requirements, as each cycle provides opportunities for feedback and adjustments. However, this approach can require more time and resources to complete, as there may be a need to rework earlier cycles to incorporate changes.

5. **Agile Model**: Agile is an iterative and incremental approach to software development that emphasizes collaboration, flexibility, and responsiveness to change. The Agile Manifesto highlights values such as working software over comprehensive documentation, individual interactions over tools, responding to change over following a plan, and customer collaboration over contract negotiation. Agile teams typically work in short sprint cycles, delivering functional features incrementally, and continuously seeking feedback from customers and stakeholders. Popular Agile methodologies include Scrum, Kanban, Lean, Crystal, Extreme Programming (XP), and Feature Driven Development (FDD).

6. **Rapid Prototyping Model**: The rapid prototyping model involves quickly building a prototype version of the software to demonstrate functionality, gather feedback, and validate assumptions early in the development process. Multiple iterations of the prototype are developed, tested, and improved upon until the desired level of maturity is reached. The goal is to reduce the overall development time and minimize the risk of misaligned expectations by engaging stakeholders in frequent communication and validation loops.

7. **Synchronize-and-Stabilize Life-Cycle Model**: Also known as concurrent engineering or parallel synchronous development, the synchronize-and-stabilize model focuses on overlapping various phases of the software development process to accelerate time-to-market and improve efficiency. Teams coordinate their efforts through regular meetings and milestones to keep pace with interdependent tasks. The stabilization phase occurs near the end of the development cycle, allowing teams to focus on resolving any remaining issues and preparing the software for release.

8. **Open-Source Life-Cycle Model:** The open-source life-cycle model is a collaborative approach to software development that leverages community contributions to build, maintain, and enhance software systems. Developers follow transparent and public processes, inviting external participation and review. The model encourages sharing of knowledge, resources, and expertise across a wide range of contributors, often resulting in faster innovation, lower costs, and higher-quality outcomes compared to proprietary alternatives. Examples of popular open-source projects include Linux operating system, Apache web server, MySQL database, and Mozilla Firefox browser.

Waterfall Model

Advantages:

1. **Simplicity and Easy Management**: The linear and sequential nature makes it easy to understand and manage.
2. **Clear Milestones and Documentation**: Each phase must be completed before the next begins, ensuring comprehensive documentation and clear milestones.
3. **Predictable Timeline and Budget**: The structure allows for better prediction of timelines and budgets.
4. **Structured Approach**: Clearly defined stages and deliverables make the process straightforward.
5. **Well-Suited for Smaller Projects**: Effective for projects with well-understood requirements and less complexity.

Disadvantages:

1. **Inflexibility to Changes**: Difficult to accommodate changes once a phase is completed.
2. **Late Testing Phase**: Issues are discovered late in the process, potentially leading to costly fixes.
3. **Poor Adaptability**: Not suitable for projects with evolving requirements.
4. **High Risk of Failure**: If initial requirements are incorrect, the entire project may fail.
5. **Limited User Involvement**: Minimal user feedback until the testing phase, which can lead to misalignment with user needs.

Spiral Model

Advantages:

1. **Risk Management**: Focuses on early identification and mitigation of risks.
2. **Flexibility and Adaptability**: Allows for changes and refinements at each iteration.
3. **Customer Feedback**: Regular customer involvement ensures the product meets user expectations.
4. **Incremental Development**: Enables partial product releases, providing early insights and feedback.
5. **Comprehensive Coverage**: Combines design, prototyping, and testing in stages, ensuring thorough development.

Disadvantages:

1. **Complexity**: More complex and difficult to manage than linear models.
2. **High Cost**: Iterative nature can increase costs and extend timelines.
3. **Dependency on Risk Analysis**: Effectiveness depends on the accuracy of risk analysis.
4. **Requires Expertise**: Demands highly skilled and experienced team members.
5. **Difficult to Predict**: Challenging to estimate time and budget accurately due to iterative cycles.

Unified Process Model (UPM)

Advantages:

1. **Iterative and Incremental**: Allows for continuous refinement and delivery of the product.
2. **Risk Mitigation**: Early identification and management of risks.
3. **Customizable**: Can be tailored to fit the specific needs of the project.

4. **Comprehensive Documentation**: Ensures thorough documentation and clarity.
5. **Focus on Quality**: Emphasizes best practices and quality assurance.

Disadvantages:

1. **Complexity**: Can be complex to implement and manage.
2. **High Resource Requirement**: Requires significant resources in terms of time and skilled personnel.
3. **Steep Learning Curve**: Team members need to understand and adopt a new methodology.
4. **Overhead Costs**: Detailed documentation and iterative cycles can increase costs.
5. **Potential for Over-Engineering**: Risk of producing more features than necessary, leading to bloat.

Iterative-and-Incremental Model

Advantages:

1. **Flexibility**: Can accommodate changing requirements and feedback throughout development.
2. **Early Delivery**: Allows for early delivery of partial working software, providing early insights.
3. **Risk Reduction**: Regular iterations help identify and mitigate risks early.
4. **Improved Quality**: Continuous testing and evaluation enhance overall quality.
5. **User Feedback**: Frequent cycles of feedback ensure alignment with user needs.

Disadvantages:

1. **Resource Intensive**: Requires more time and resources due to repeated cycles.
2. **Complex Management**: Managing multiple iterations and changes can be complex.
3. **Scope Creep**: Risk of scope creep due to ongoing changes and additions.
4. **Incomplete Systems**: Early iterations may not be fully functional, leading to user dissatisfaction.
5. **Potential Delays**: Frequent changes and iterations can lead to delays in final delivery.

Agile Model

Advantages:

1. **Flexibility and Responsiveness**: Highly adaptable to changing requirements and customer needs.
2. **Continuous Feedback**: Regular feedback from stakeholders ensures the product meets user expectations.
3. **Early and Frequent Delivery**: Delivers functional software early and frequently, providing early value.
4. **Enhanced Collaboration**: Emphasizes collaboration and communication among team members and stakeholders.
5. **Improved Quality**: Continuous testing and iteration improve overall software quality.

Disadvantages:

1. **Scope Creep**: Flexibility can lead to uncontrolled scope creep if not managed properly.
2. **Demanding on Teams**: Requires highly skilled, committed, and collaborative teams.
3. **Less Predictable**: Difficult to predict timelines, costs, and outcomes accurately.

4. **Documentation Shortcomings**: Focus on working software over documentation can lead to insufficient documentation.
5. **Scaling Challenges**: Can be challenging to scale for large, complex projects.

Rapid Prototyping Model

Advantages:

1. **Quick Feedback**: Early prototypes provide immediate feedback and validation.
2. **Risk Reduction**: Identifies potential issues and misunderstandings early in the process.
3. **Stakeholder Engagement**: Engages stakeholders early and often, ensuring alignment with their needs.
4. **Enhanced Understanding**: Helps clarify requirements and improve understanding among team members.
5. **Reduced Time to Market**: Speeds up the development process by quickly iterating on prototypes.

Disadvantages:

- **Limited Functionality**: Initial prototypes may lack full functionality, leading to misunderstandings.
- **High Resource Demand**: Frequent iterations can consume significant resources.
- **Potential for Incomplete Documentation**: Focus on prototypes can result in inadequate documentation.
- **Risk of Incomplete Solutions**: Risk of moving forward with incomplete or inaccurate solutions.
- **Overemphasis on Design**: May focus too much on design and not enough on scalability and maintainability.

- **Waterfall vs Agile Models**: Waterfall model works well for small to medium-sized projects where there is little uncertainty about the final product and its requirements. On the other hand, agile methods like Scrum or XP work well for larger, more complex projects with evolving requirements. They provide greater visibility into progress, allow for quicker response to changing requirements, and encourage collaboration between developers and stakeholders.
- **Spiral vs Waterfall:** The spiral model allows for incremental delivery and continuous feedback from users, making it suitable for large and complex projects involving significant risks. This makes it particularly useful when developing new products or technologies where the requirements may not be fully understood at the outset. Compared to the waterfall model, the spiral model offers more opportunities for iteration, refinement, and course correction during the development process.
- **Unified Process vs Other Models:** UPM offers a balanced approach that incorporates many of the benefits of both traditional plan-based approaches (e.g., waterfall) and agile methodologies. By providing guidelines for each stage of the software development process, UPM helps ensure thoroughness and completeness of the deliverables. At the same time, UPM's emphasis on iterative development and continuous improvement aligns closely with modern agile principles. Overall, UPM can be seen as a flexible framework that can be adapted to different project types and sizes, offering a balance between structure and adaptability

Software Development Life Cycle (SDLC): SDLC is a process used by the software industry to design, develop, and test high-quality software. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

1. **Planning and Requirement Analysis**: Define the project's basic outline, obtain customer inputs, and perform requirement analysis. Create a preliminary design based on available information.
2. **Defining Requirements**: Detail all the software requirements in the form of a Software Requirement Specification (SRS) document. Obtain approvals from relevant parties.
3. **Designing Architecture**: Use the SRS as a reference to create multiple architectural designs. Assess them against various criteria and select the most practical option.
4. **Developing Product**: Build the software based on the chosen design using specific programming codes and standard programming tools. Follow established protocols and conventions.
5. **Product Testing and Integration**: Thoroughly test the software to detect and fix bugs. Ensure that the product adheres to the quality standards outlined in the SRS.
6. **Deployment and Maintenance of Products**: Release the software in phases, monitor its performance, address customer feedback, and conduct post-release maintenance.

Object-oriented programming (OOP) is a programming paradigm that organizes software design around objects, which are instances of classes. OOP focuses on modelling real-world entities and their interactions by encapsulating data (attributes) and behaviour (methods) into objects. The key principles and concepts of OOP include:

1. **Encapsulation**: Encapsulation refers to the bundling of data and methods within an object. It allows for data hiding, ensuring that the internal state of an object can only be accessed and modified through defined methods. Encapsulation promotes information hiding and improves the maintainability and reusability of code.
2. **Inheritance**: Inheritance enables the creation of new classes (derived or child classes) based on existing classes (base or parent classes). The derived class inherits the attributes and methods of the base class, allowing for code reuse and promoting the concept of "is-a" relationship. Inheritance facilitates the modelling of hierarchical relationships and promotes modularity and extensibility.
3. **Polymorphism**: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It provides the ability to perform a single action in different ways based on the context or the type of objects involved. Polymorphism promotes code flexibility, reusability, and modularity.
4. **Abstraction**: Abstraction focuses on representing essential features of an object while hiding unnecessary details. It allows for the creation of abstract classes and interfaces, defining common properties and behaviours that can be inherited by subclasses. Abstraction facilitates the conceptualization and organization of complex systems and enhances code maintainability.

Post maintenance delivery

Post-delivery maintenance refers to the activities and processes that take place after a product or project has been delivered to ensure its continued performance, reliability, and user satisfaction. This maintenance phase is crucial for the longevity and success of the product or system.

1. **A fault needs correcting**, whether an analysis fault, design fault, coding fault, documentation fault, or any other type of fault. This is termed corrective maintenance.
2. **In perfective maintenance**, a change is made to the code to improve the effectiveness of the product. For instance, the client may wish additional functionality or request that the product be modified so that it runs faster. Improving the maintainability of a product is another example of perfective maintenance.
3. **In adaptive maintenance**, a change is made to the product to react to a change in the environment in which the product operates. For example, a product almost certainly has to be modified if it is ported to a new compiler, operating system, or hardware. With each change to the tax code, a product that prepares tax returns has to be modified accordingly. When the U.S. Postal Service introduced nine-digit ZIP codes in 1981, products that had allowed for only five digit ZIP codes had to be changed. Adaptive maintenance is not requested by a client; instead, it is externally imposed on the client

Testing

Non-Execution-Based Testing

Non-execution-based testing refers to the methods of software testing that do not involve running the actual software or its code. Instead, these testing techniques focus on analysing and evaluating the software's documentation, design, and code without execution.

- Non-execution-based testing includes reviews and inspections.
- Walkthroughs involve carefully reading through documents with a team, identifying faults.
- Inspections follow a formal process with specific steps and use checklists to find faults.
- Reviews detect faults early in the process, reducing costs and improving quality.
- Metrics like inspection rate, fault density, and fault detection rate measure the effectiveness of inspections.

Execution-Based Testing:

This involves running a product with selected inputs to infer its behavioural properties. However, there are challenges, such as uncertainty about the environment and lack of control over inputs, especially in real-time systems like avionics software.

Properties to Test:

- **Utility**: Tests whether the product meets user needs and is easy to use.
- **Reliability:** Measures frequency and criticality of failures and their impact.
- **Robustness:** Tests the product's ability to handle various conditions and inputs.
- **Performance**: Evaluates how well the product meets constraints like response time or space requirements.
- **Correctness**: Ensures the product satisfies output specifications under permitted conditions.

- **Correctness Proofs:** Mathematical techniques (verification) can prove a product's correctness. Despite challenges like mathematical complexity, cost, and difficulty, they've been successfully applied in critical systems.

Benefits of testing

- **Customer satisfaction:** The primary objective of the owner of the products is to offer the best satisfaction to the customers. The reason why it is necessary to opt for software testing is that it offers the prerequisite and perfect user experience.
- **Low Failure:** Failure of an application impacts its working and brand value. Software testing helps in knowing the cases where a particular application is most likely to fail.
- **Reliable Product:** A product is reliable only when it performs as per user requirements and can build customer trust. Software testing increases the reliability of an application by testing the application with security testing, performance testing, and other testing techniques.
- **Cost Effective:** If an application works without any fault and with low maintenance will save a big amount for the owner. Software testing helps in early defect detection and fixing them to make a more successful and good return application.
- **Security**: The testing team uses security testing to identify defects and the development team tries to cover the application with multiple security layers. Thus, software testing is a must to maintain the security of an application

1. Software Engineering
    - Is a discipline whose aim is the production of fault-free software delivered on time, within budget, and satisfying the user's needs.
2. Web engineering
    - Is a discipline whose aim is the production of fault-free Web software delivered on time, within budget, and satisfying the user's needs.
3. Stepwise refinement
    - Can be defined as a means to postpone decisions on details until as late as possible to concentrate on the important issues.
4. Cost–benefit analysis (
    - One way of determining whether a possible course of action would be profitable is to compare estimated future benefits against projected future costs.

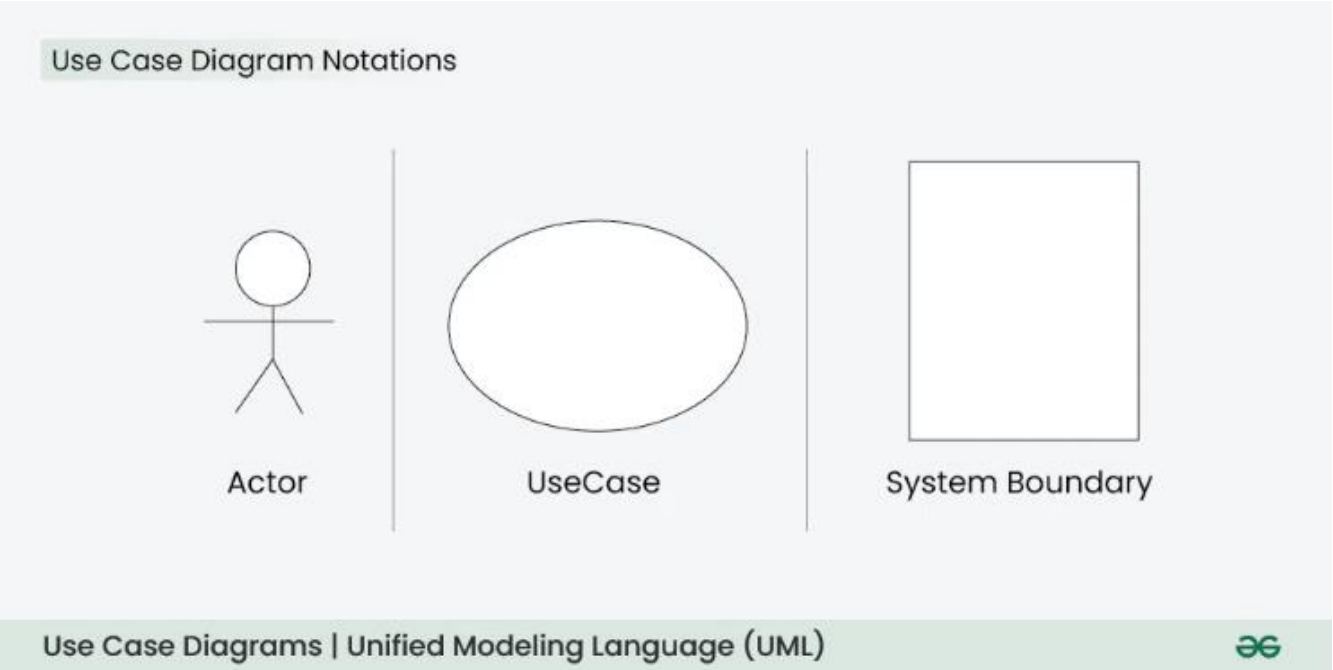
Classical life-cycle model

1. **Requirements Phase**: Identify and refine the concept, eliciting the client's needs.
2. **Analysis Phase**: Analyse and document the client's requirements, creating a detailed specification.
3. **Design Phase**: Break down the product into modules (architectural design) and design each module in detail.
4. **Implementation Phase**: Code and test components separately, then integrate them. Client conducts acceptance testing before installation.
5. **Post-Delivery Maintenance**: Maintain the product after installation, including corrective maintenance (fixing faults) and enhancement (updating specifications).
6. **Retirement**: Remove the product from service when it's no longer useful to the client.

The responsibilities of the software project manager

Software project manager is engaged with software management activities. He / She is responsible for project planning, monitoring the progress, communication among stakeholders, managing risks and resources, smooth execution of development and delivering the project within time, cost and quality constraints.

Quality Assurance monitors to check if proper process is followed while software developing the software. Quality Control deals with maintaining the quality of software product.



Use case example:

Using your knowledge of how an Online Banking System works, develop/design use cases for the online banking system. Clearly explain using scenarios/diagrams.

- A Customer is required to create an account to avail services offered by Bank. Bank verifies detail and creates new account for each new customer. Each customer is an actor for the Use-Case Diagram and the functionality offered by Online Banking System to Add Account is Use-Case.
- Each customer can check the balance in bank account and initiate request to transfer an account across distinct branches of Bank. Cashier is an employee at bank who supports service to the customer.

- A customer can execute cash transactions where the customer must either add cash value to bank account or withdraw cash from account. Either of two or both that is credit as well as debit cash, might be executed to successfully execute one or multiple transactions.
- After each successful transaction customer might or might not want to get details for action. Manager can check interest value for each account corresponding to transaction to ensure and authenticate details.
- A customer can also request loan from bank where customer must add request for loan with the appropriate details.
- The type of loan in accordance with purpose or the need for loan and term or duration to pay back the loan must be provided by customer.
- The manager of each branch of bank has choice to either accept or approve loan to initiate process further or just reject request for loan based on terms and conditions.
- The record for each employee of bank is maintained by bank and bank manages all employees of each branch of bank. The manager of each branch has choice to offer bonus to employees. Note here that each employee is paid as part of management of staff but promotion or bonus might or might not be offered certainly to each employee.