SE Mod 1

Software engineering: Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements. It is an engineering field that is concerned with all aspects of production from early stages of system specifications to maintaining the system after it has gone to use

Importance of SE:
- Reduces complexity
- Handling big projects
- Minimise software cost
- Decrease time
- Effectiveness
- Reliable software

SDLC: Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
SDLC phases :
- Planning and requirements analysis
- Defining requirements
- Designing the product architecture
- Build the product
- Test the product
- Deploy in market and maintain

Software models - models that follow a series of steps to ensure successful product development

Waterfall model - most basic, simple, sequential model, used when requirements are not changing frequently and project is short and simple. Adv- each phase must be completed before next, suites for small projects, simple n easy to understand. Disadvantage- error can be fixed only in that phase, no customer feedback
Phases -
- Requirement analysis
- System design
- Implementation
- System design testing

- System deployment
- System maintenance

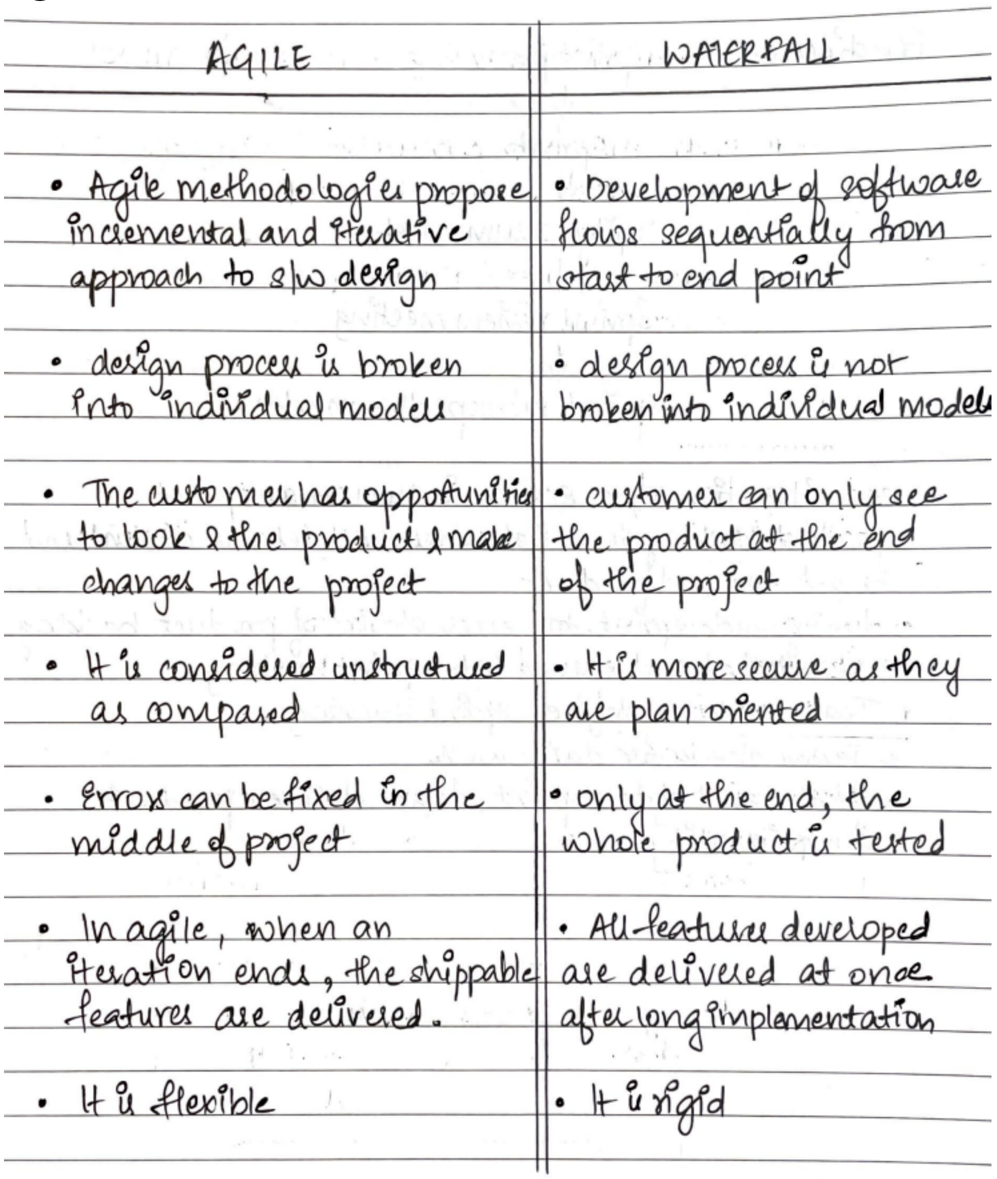
Spiral model - combination of iterative and the waterfall model, high emphasis on risk analysis,
4 phases and software repeatedly goes through the phases, at the end of each spiral the product is deployed in the market. Adv- flexible in req, large projects, risk handling at each phase. Disadvantage- complex, expensive
Phases-

- Requirement analysis
- Design
- Coding
- Testing and risk analysis

Agile Model - comb of iterative and incremental model, emphasis on process adaptability and customer satisfaction by rapid delivery, divides tasks into time boxes to provide a specific functionality for the release, final build contains all attributes, believes every project needs to be handled differently thus it follows adaptive approach. Adv- realistic, suits changing req. disadvantage- not for complex dependencies, diff to estimate cost and resources needed.
Phases -

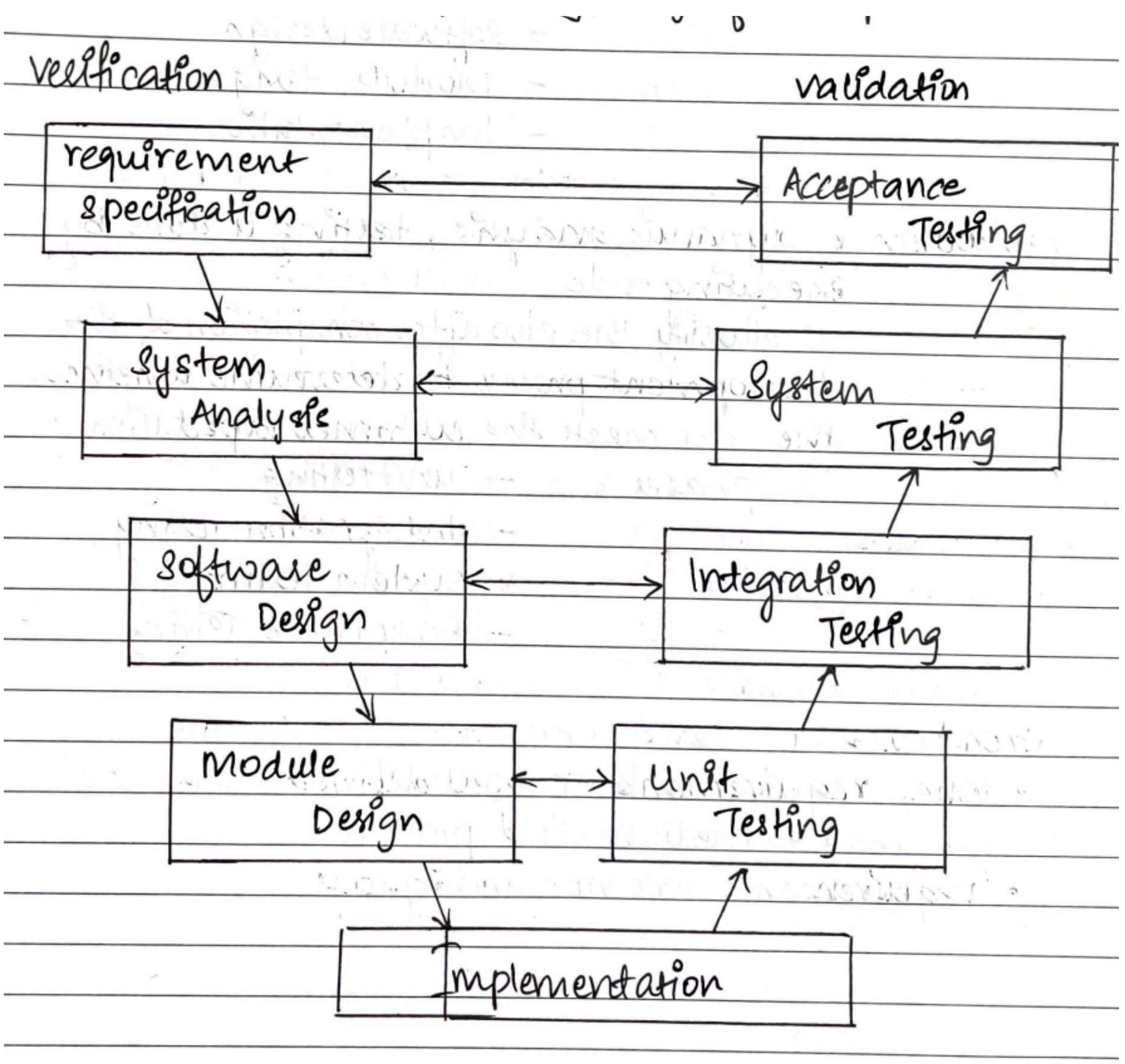
- Planning
- Requirement analysis
- Designing
- Building
- Testing

Agile vs waterfall



AGILE	WATERFALL
• Agile methodologies propose incremental and iterative approach to s/w design	• Development of software flows sequentially from start to end point
• design process is broken into individual models	• design process is not broken into individual models
• The customer has opportunities to look & the product & make changes to the project	• customer can only see the product at the end of the project
• It is considered unstructured as compared	• It is more secure as they are plan oriented
• Errors can be fixed in the middle of project	• only at the end, the whole product is tested
• In agile, when an iteration ends, the shippable features are delivered.	• All features developed are delivered at once after long implementation
• It is flexible	• It is rigid

V Model - verification and validation model, sequential, testing is planned in parallel with corresponding stage. Verification: static analysis without executing the code, evaluate processes, validation: dynamic analysis, testing is done by executing the code
Phases-



RAD Model- rapid app development, based on prototyping without any specific planing, the functional modules are developed in parallel as prototypes and integrated to make complete product, focuses on fast delivery
Phases-

- Business modelling
- Data modelling
- Process modelling
- Application generation
- Testing and turnover

DevOps - comb of software development and operations, methodology that aims to integrate work of development and operation teams by facilitating a culture of collaboration and shared responsibility, team empowerment

CMM- Capability maturity model, framework used to analyse the approach followed by an organisation to develop a software products, develop and refine the processes, guidelines to further enhance

SE Mod 21

SRS - It is a document that outlines the detailed requirements for a software system. An SRS is typically created by a software developer or a team of developers, and it serves as a reference for the entire development process. The purpose of an SRS is to clearly communicate the requirements of a software system to all stakeholders, including developers, testers, and users. It outlines the functional (login, shopping cart, payment, logout, register) and non-functional (security, reliability, performance) requirements of the system, as well as any constraints or assumptions that must be taken into consideration during development.

DFD : DFD stands for Data Flow Diagram. It is a graphical representation of the flow of data within a system. A DFD is used to visualize how data moves through various processes within a system, from inputs to outputs. DFDs are often used in the early stages of software development to help identify and clarify the requirements and goals of a system. 4 symbols- external entity, process, data, data flow
0-level DFD - represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Level 1 - main objectives, level 2 - deeper parts of level 1

Use case diagram - A use case diagram is a type of diagram used in software development to visually represent the different ways that users might interact with a system. Use case diagrams are typically used to model the functional requirements of a system from the perspective of the users.In a use case diagram, the actors who interact with the system are represented as stick figures, while the use cases (i.e., the various functions or tasks that the users perform) are represented as ovals. Arrows are used to indicate the flow of information or actions between the actors and the use cases.
Used to understand functional requirements of system . 3 types of relationships exist among use cases -

- Include relationship
- Extend relationship
- Use case generalisation

Requirements modeling is the process of identifying the requirements this software solution must meet in order to be successful.
Scenario-based Model :Using a scenario-based approach, system is described from user's point of view.
The behavioral model indicates how software will respond to external events or stimuli.

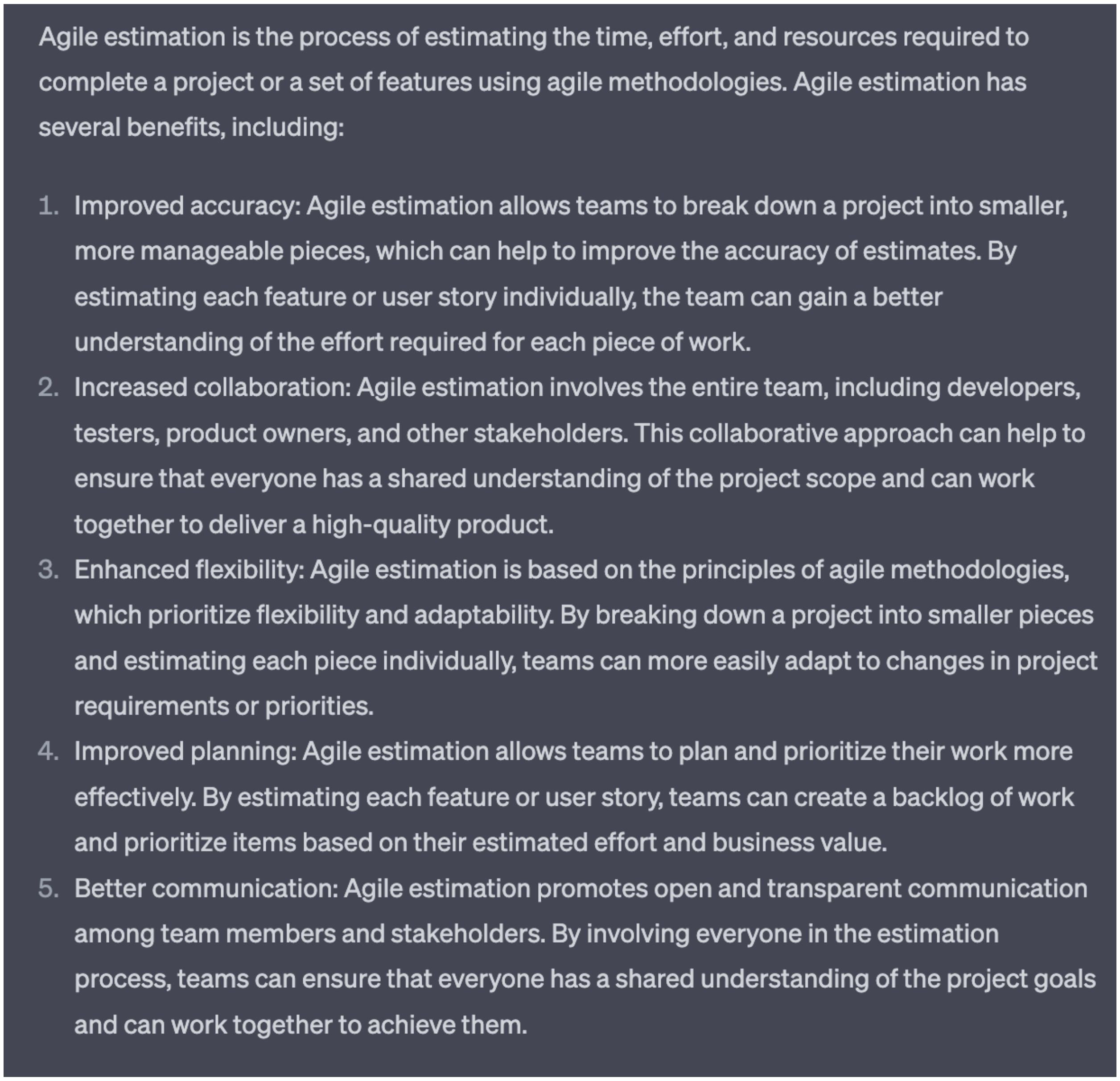
• **Class diagram:** A class diagram is a type of structural diagram that shows the classes, interfaces, associations, and other objects in a system, along with their attributes and methods. Class diagrams are used to model the static structure of a system, and are often used as a starting point for software design.
• **State/activity diagram:** A state/activity diagram is a type of behavioral diagram that shows the different states and activities of a system or software application. It is used to model the dynamic behavior of a system, and can be used to visualize the sequence of events that occur during the execution of a software program.
• **Sequence diagram:** A sequence diagram is a type of interaction diagram that shows the sequence of interactions between objects in a system. It is used to model the dynamic behavior of a system, and can be used to visualize the flow of control between different objects and components in a software application.

A Gantt chart is a type of bar chart used in project management to visualize a project schedule. It is used to show the start and end dates of individual tasks or activities in a project, as well as their dependencies and overall progress.

SE Mod 3

Project estimation -Software project estimation is the process of predicting the time, cost, and resources required to complete a software development project.
LOC : as Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.
FP : It is a tool to help users discover the benefit of an application package to their organization by counting functions that specifically match their requirements.

COCOMO Model: The COCOMO (Constructive Cost Model) The model is based on the premise that the cost of developing software is directly proportional to the size of the software product being developed. The output of the COCOMO model is an estimate of the effort required to complete the project, expressed in person-months or person-years.



Agile estimation is the process of estimating the time, effort, and resources required to complete a project or a set of features using agile methodologies. Agile estimation has several benefits, including:

1. Improved accuracy: Agile estimation allows teams to break down a project into smaller, more manageable pieces, which can help to improve the accuracy of estimates. By estimating each feature or user story individually, the team can gain a better understanding of the effort required for each piece of work.
2. Increased collaboration: Agile estimation involves the entire team, including developers, testers, product owners, and other stakeholders. This collaborative approach can help to ensure that everyone has a shared understanding of the project scope and can work together to deliver a high-quality product.
3. Enhanced flexibility: Agile estimation is based on the principles of agile methodologies, which prioritize flexibility and adaptability. By breaking down a project into smaller pieces and estimating each piece individually, teams can more easily adapt to changes in project requirements or priorities.
4. Improved planning: Agile estimation allows teams to plan and prioritize their work more effectively. By estimating each feature or user story, teams can create a backlog of work and prioritize items based on their estimated effort and business value.
5. Better communication: Agile estimation promotes open and transparent communication among team members and stakeholders. By involving everyone in the estimation process, teams can ensure that everyone has a shared understanding of the project goals and can work together to achieve them.

CPM stands for Critical Path Method, which is a project management technique used to plan and manage complex projects. CPM involves breaking down a project into individual tasks or activities, and then identifying the sequence of tasks required to complete the project.

SE mod 4

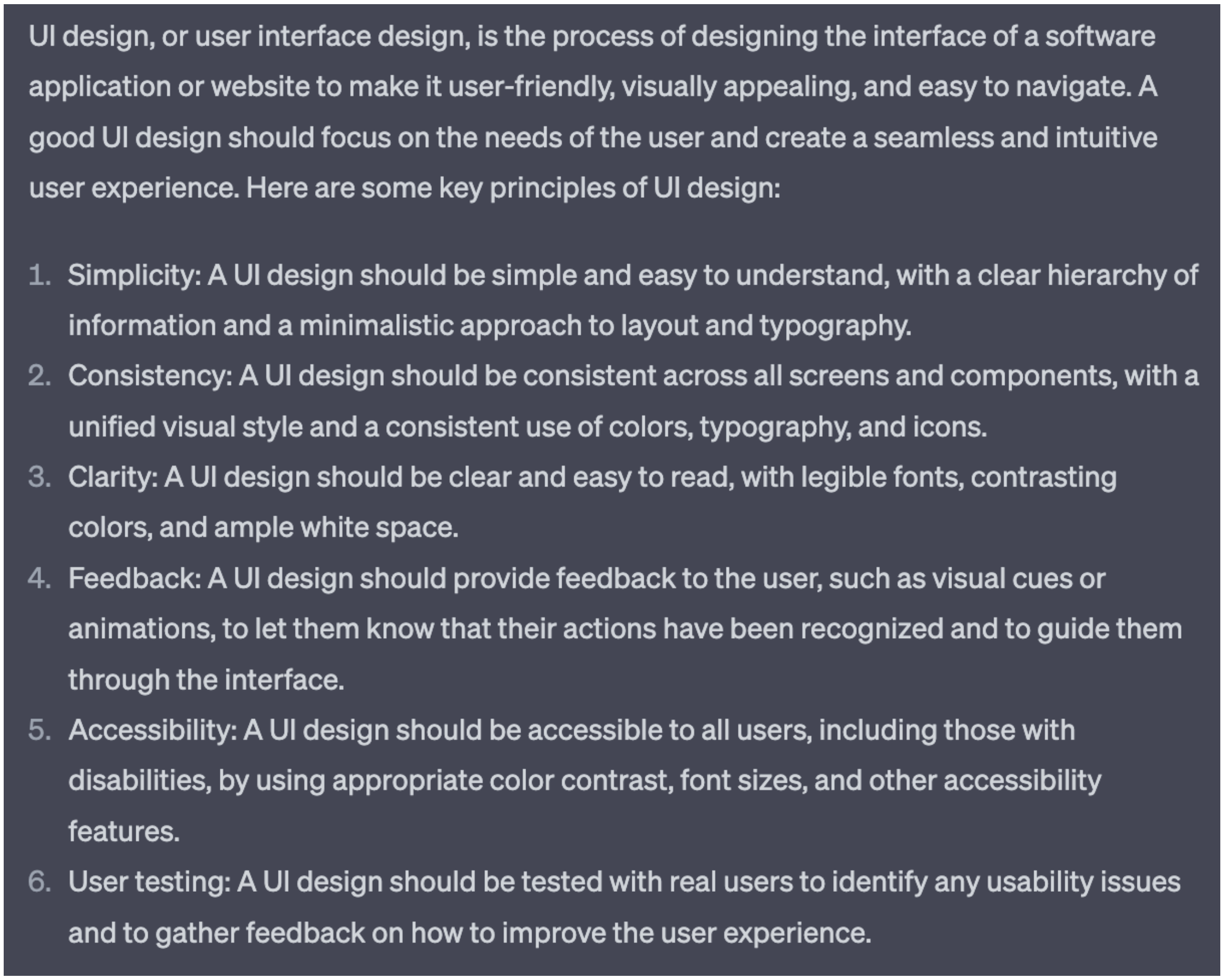
Characteristics of a good design:

- Maintainability
- Scalability
- Usability
- Performance
- Security
- Flexibility

Cohesion and Coupling: Cohesion and coupling are two important concepts in software design that describe how modules or components within a system are related to each other.

Cohesion refers to the degree to which the elements of a module or component work together to achieve a single, well-defined purpose or functionality. A module with high cohesion is one where all its elements are related and contribute to the same purpose or functionality, while a module with low cohesion contains unrelated elements that perform different tasks.High cohesion is generally desirable in software design because it leads to better maintainability and reusability, as well as reduced complexity and fewer errors.

Coupling on the other hand, refers to the degree to which one module or component depends on another. High coupling means that one module is closely dependent on another, while low coupling means that modules are relatively independent and can be changed or replaced without affecting other parts of the system. Low coupling is generally desirable in software design because it leads to greater flexibility, maintainability, and reusability. High coupling can lead to more complex and difficult-to-maintain code, as well as a greater likelihood of errors and failures.


UI design, or user interface design, is the process of designing the interface of a software application or website to make it user-friendly, visually appealing, and easy to navigate. A good UI design should focus on the needs of the user and create a seamless and intuitive user experience. Here are some key principles of UI design:

1. Simplicity: A UI design should be simple and easy to understand, with a clear hierarchy of information and a minimalistic approach to layout and typography.
2. Consistency: A UI design should be consistent across all screens and components, with a unified visual style and a consistent use of colors, typography, and icons.
3. Clarity: A UI design should be clear and easy to read, with legible fonts, contrasting colors, and ample white space.
4. Feedback: A UI design should provide feedback to the user, such as visual cues or animations, to let them know that their actions have been recognized and to guide them through the interface.
5. Accessibility: A UI design should be accessible to all users, including those with disabilities, by using appropriate color contrast, font sizes, and other accessibility features.
6. User testing: A UI design should be tested with real users to identify any usability issues and to gather feedback on how to improve the user experience.

SE Mod 5

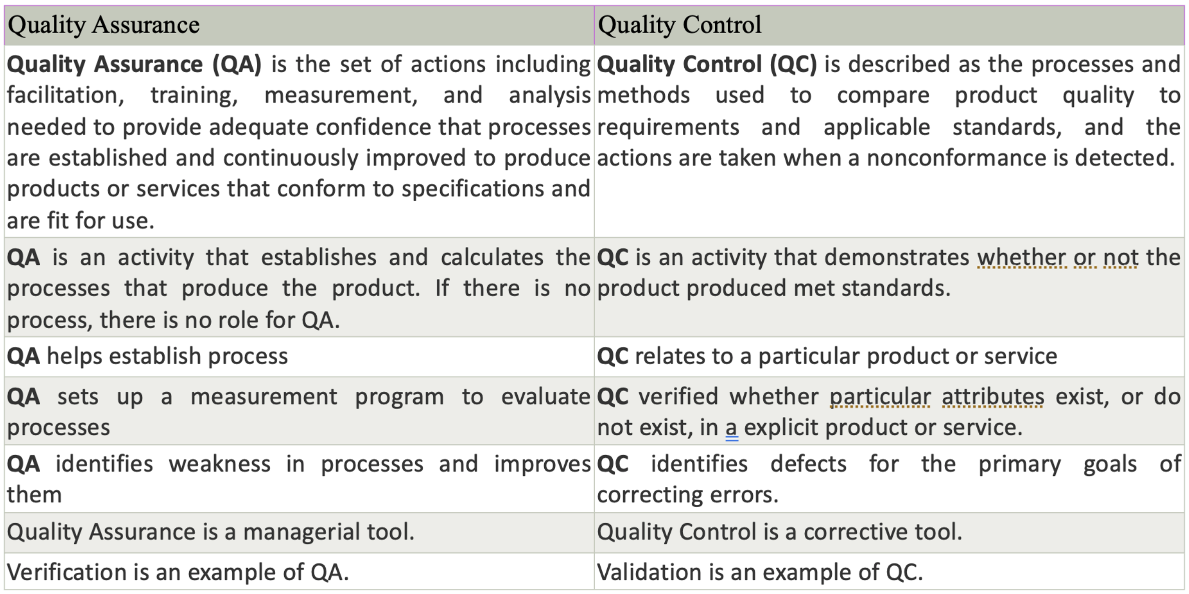
Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty.
Types of risk- project, technical, business, market, known, unpredictable, predictable

Risk Mitigation: It is an activity used to avoid problems (Risk Avoidance).
Risk Monitoring: It is an activity used for project tracking

Difference between SCM and Software Support (Maintenance)

- Support is a set of software engineering activities that occur after software has been delivered to the customer and put into operation.
- Software configuration management is a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.

Quality Assurance: Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully. Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements. A set of activities designed to calculate the process by which the products are developed or manufactured.

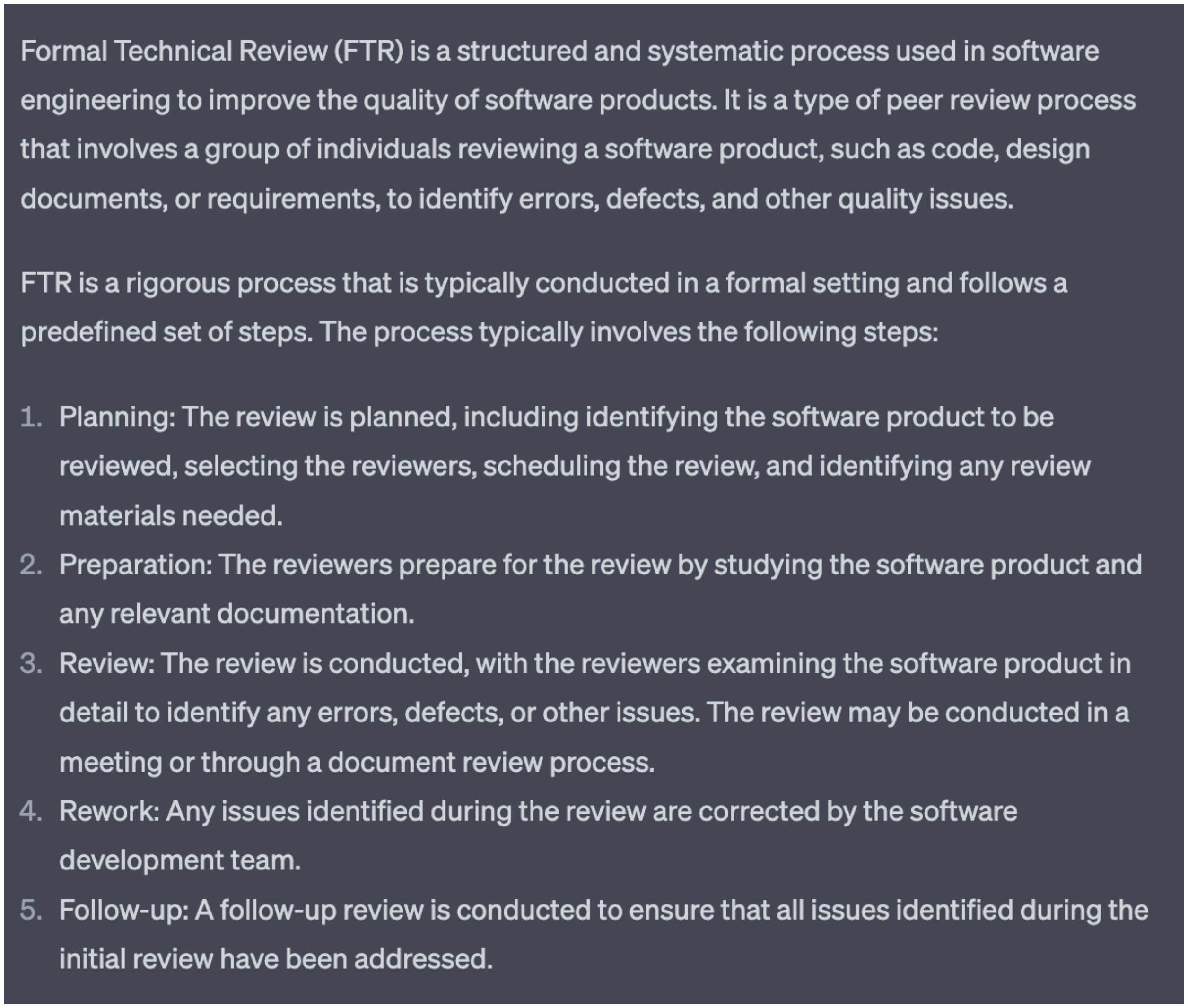


Quality Assurance	Quality Control
Quality Assurance (QA) is the set of actions including facilitation, training, measurement, and analysis needed to provide adequate confidence that processes are established and continuously improved to produce products or services that conform to specifications and are fit for use.	**Quality Control (QC)** is described as the processes and methods used to compare product quality to requirements and applicable standards, and the actions are taken when a nonconformance is detected.
QA is an activity that establishes and calculates the processes that produce the product. If there is no process, there is no role for QA.	**QC** is an activity that demonstrates whether or not the product produced met standards.
QA helps establish process	**QC** relates to a particular product or service
QA sets up a measurement program to evaluate processes	**QC** verified whether particular attributes exist, or do not exist, in a explicit product or service.
QA identifies weakness in processes and improves them	**QC** identifies defects for the primary goals of correcting errors.
Quality Assurance is a managerial tool.	Quality Control is a corrective tool.
Verification is an example of QA.	Validation is an example of QC.

Six Sigma is the process of improving the quality of the output by identifying and eliminating the cause of defects and reduce variability in manufacturing and business processes.

RMMM is a software engineering model that stands for Risk Mitigation, Monitoring, and Management. It is a risk-based approach to software development that focuses on identifying, assessing, and mitigating risks throughout the software development life cycle.

- Risk mitigation: In this phase, strategies are developed to mitigate the identified risks. This may involve developing contingency plans, implementing risk controls, or redesigning aspects of the software system to reduce the likelihood or impact of a risk.
- Risk monitoring: In this phase, the identified risks are continually monitored throughout the software development life cycle to ensure that the risk mitigation strategies are effective. This may involve tracking the progress of risk mitigation activities, updating risk assessments as new information becomes available, and communicating risk information to stakeholders as needed.
- Risk management: In this phase, the overall risk management process is reviewed and refined to ensure that it remains effective and responsive to changing project needs and circumstances.



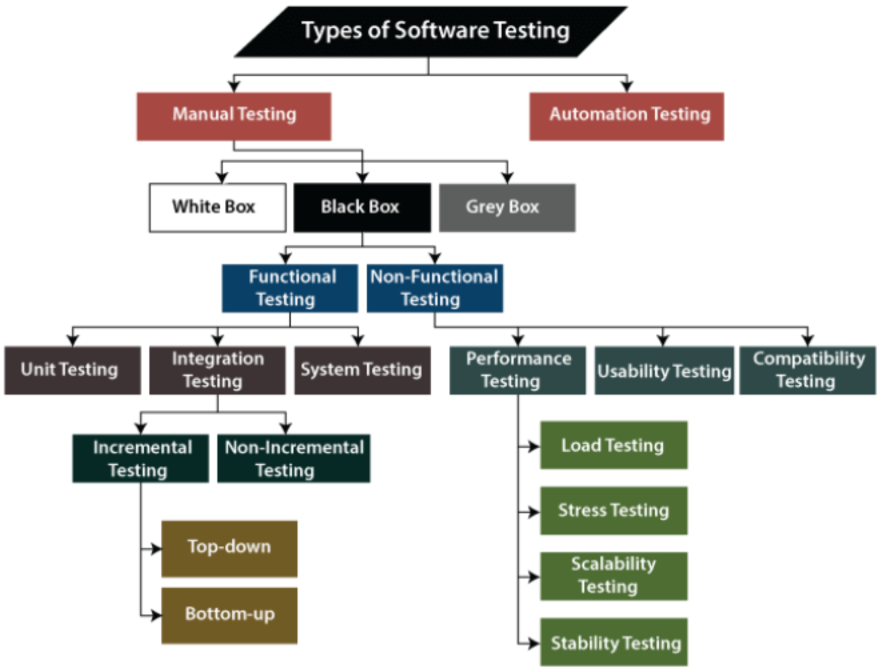
Formal Technical Review (FTR) is a structured and systematic process used in software engineering to improve the quality of software products. It is a type of peer review process that involves a group of individuals reviewing a software product, such as code, design documents, or requirements, to identify errors, defects, and other quality issues.

FTR is a rigorous process that is typically conducted in a formal setting and follows a predefined set of steps. The process typically involves the following steps:

1. Planning: The review is planned, including identifying the software product to be reviewed, selecting the reviewers, scheduling the review, and identifying any review materials needed.
2. Preparation: The reviewers prepare for the review by studying the software product and any relevant documentation.
3. Review: The review is conducted, with the reviewers examining the software product in detail to identify any errors, defects, or other issues. The review may be conducted in a meeting or through a document review process.
4. Rework: Any issues identified during the review are corrected by the software development team.
5. Follow-up: A follow-up review is conducted to ensure that all issues identified during the initial review have been addressed.

SE Mod 6

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance.



- **Unit testing:** This is the process of testing individual units or modules of code to ensure that they are working correctly. It typically involves writing test cases for each unit and verifying that they pass those tests. Unit testing is usually done by developers during the development process and can help to catch bugs early on.
- **Integration testing:** This is the process of testing how individual modules or components of a system work together to ensure that they are integrated correctly. It typically involves testing the interfaces between modules and verifying that they work as expected. Integration testing is usually done after unit testing and can help to catch issues with how different parts of the system interact.
- **Verification testing:** This is the process of testing that the software product meets its requirements and specifications. It typically involves testing that the product behaves as expected under different scenarios and inputs. Verification testing is usually done by QA testers to ensure that the software meets the requirements set out in the design and development phases.

- **Validation testing:** This is the process of testing that the software product meets the needs and expectations of its users. It typically involves testing the product in a real-world environment to ensure that it works as intended and meets user needs. Validation testing is usually done by end-users or by specialized testers.
- **System testing:** This is the process of testing the entire system as a whole to ensure that all its components work together correctly. It typically involves testing the system under different scenarios and configurations to ensure that it performs as expected. System testing is usually done after integration testing and can help to catch issues that only arise when all parts of the system are working together.

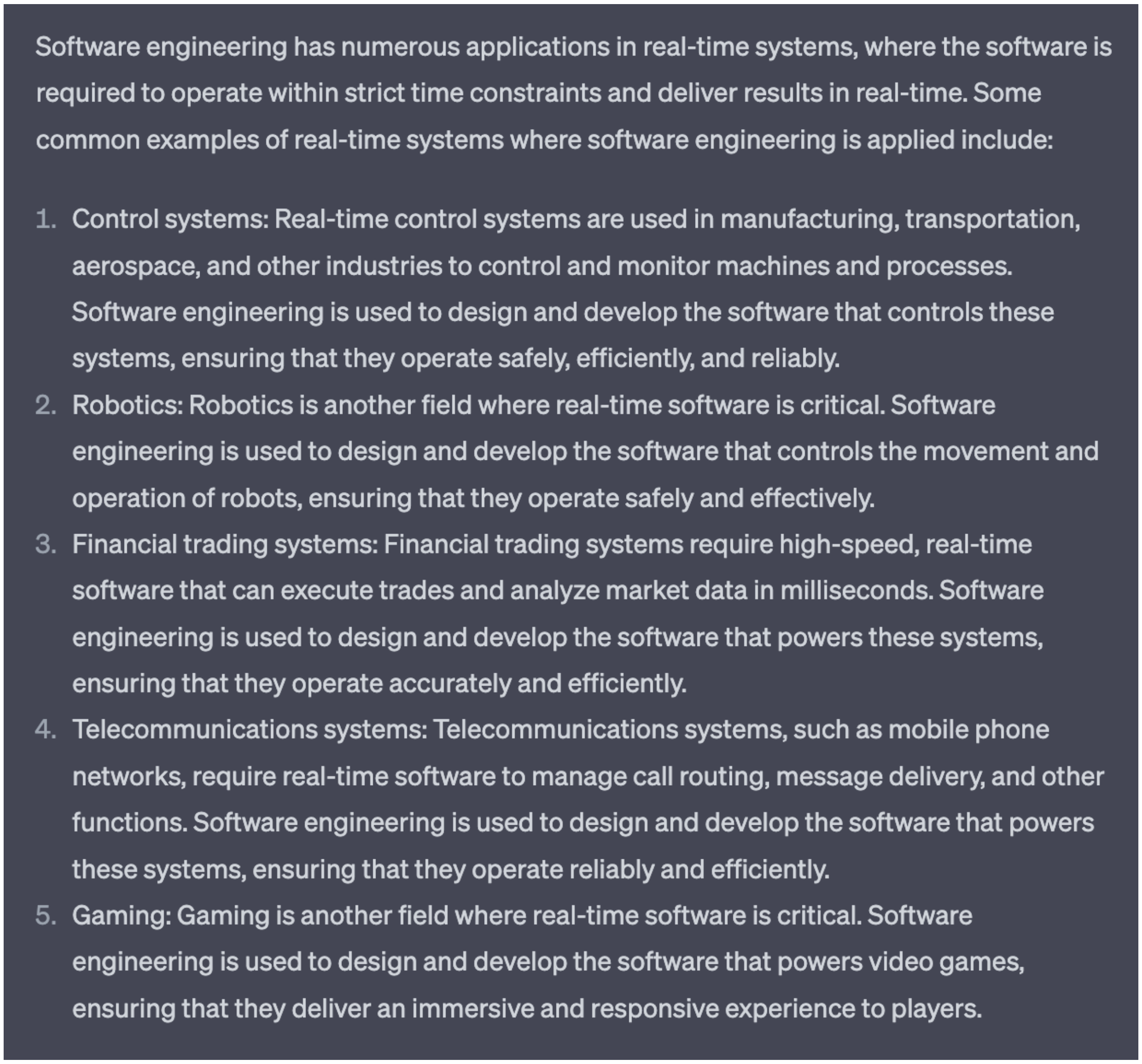
White-box testing is a type of software testing that focuses on the internal structure and implementation of a software system. It is also known as structural testing or code-based testing.
In white-box testing, the tester has access to the source code of the software being tested and can examine the internal workings of the system. The goal of white-box testing is to ensure that the software functions correctly according to its design and implementation.

Black-box testing is a type of software testing that focuses on the external behavior of a software system, without examining its internal workings or implementation. It is also known as functional testing or specification-based testing.
In black-box testing, the tester does not have access to the source code of the software being tested and is not concerned with how the software works internally. Instead, the tester focuses on the inputs and outputs of the software and tests its behavior according to its requirements and specifications.

Software maintenance is the process of modifying and updating software after it has been released, in order to correct defects, improve performance, or add new features. Types - Corrective,Preventative Perfective Adaptive Software Maintenance

Software re-engineering refers to the process of restructuring or transforming an existing software system in order to improve its performance, maintainability, or other quality attributes. This may involve updating the software to use modern programming languages or design patterns, redesigning the software to better meet user needs or business requirements, or improving the software's architecture or documentation.

Reverse engineering, on the other hand, is the process of analyzing an existing software system in order to understand its inner workings, design, or functionality. This may involve examining the software's code, documentation, or other artifacts to gain insights into how the software was developed or how it works.



Software engineering has numerous applications in real-time systems, where the software is required to operate within strict time constraints and deliver results in real-time. Some common examples of real-time systems where software engineering is applied include:

1. Control systems: Real-time control systems are used in manufacturing, transportation, aerospace, and other industries to control and monitor machines and processes. Software engineering is used to design and develop the software that controls these systems, ensuring that they operate safely, efficiently, and reliably.
2. Robotics: Robotics is another field where real-time software is critical. Software engineering is used to design and develop the software that controls the movement and operation of robots, ensuring that they operate safely and effectively.
3. Financial trading systems: Financial trading systems require high-speed, real-time software that can execute trades and analyze market data in milliseconds. Software engineering is used to design and develop the software that powers these systems, ensuring that they operate accurately and efficiently.
4. Telecommunications systems: Telecommunications systems, such as mobile phone networks, require real-time software to manage call routing, message delivery, and other functions. Software engineering is used to design and develop the software that powers these systems, ensuring that they operate reliably and efficiently.
5. Gaming: Gaming is another field where real-time software is critical. Software engineering is used to design and develop the software that powers video games, ensuring that they deliver an immersive and responsive experience to players.