



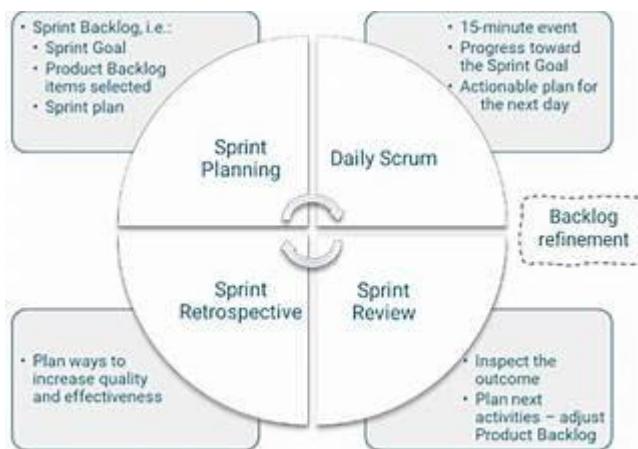
Solutions by Dare-Marvel

SOFTWARE ENGINEERING PYQS SOLUTIONS

Do all types of diagrams thoroughly 50% of paper only diagrams.

List 4 difference between Agile SCRUM and Agile XP model

<https://www.geeksforgeeks.org/difference-between-scrum-and-xp/>



Explore

Certainly! Here are four differences between Agile Scrum and Agile XP (Extreme Programming):

1. Iteration Length:

- Scrum typically works in iterations called sprints, which are usually two to four weeks long¹.
- XP teams work in shorter iterations, often one to two weeks long¹.

2. Change Management:

- Scrum teams do not allow changes to their sprints once they have started¹.
- XP is more flexible with changes and encourages them even late in the development process².

3. Engineering Practices:

- XP places a strong emphasis on engineering practices like pair programming, test-driven development (TDD), and continuous integration².
- Scrum does not prescribe specific engineering practices but focuses more on project management aspects².

4. Roles and Responsibilities:

- Scrum has defined roles such as the Product Owner, Scrum Master, and Development Team³.
- XP roles are less rigid, and the focus is more on the team as a whole with practices like collective code ownership⁴.

These differences highlight the distinct approaches each framework takes within the Agile methodology to manage and execute software development projects. Scrum is more structured in its processes, while XP is more flexible and emphasizes technical excellence.

Choose the best life cycle model for developing following software applications. Justify

- a) Well understood library automation software to link various libraries in the city. User involvement is minimum in this project.**
- b) An extremely large software that would provide, monitor and control cellular communication among its subscribers using a set of revolving satellites**

For the two software applications you've described, here are the best life cycle models and justifications for each:

a) **Library Automation Software:**

- **Best Model:** Waterfall Model
- **Justification:** Since the requirements for library automation software are well understood and unlikely to change, the Waterfall model is suitable. It allows for a systematic and sequential approach, which is ideal when user involvement is minimal. Each phase can be carefully planned and executed without the need for iterative revisions.

b) **Cellular Communication Software with Satellites:**

- **Best Model:** Agile Model
- **Justification:** Given the complexity and the need for adaptability in a project involving cellular communication and satellites, an Agile model would be best. It allows for iterative development, continuous user feedback, and the ability to respond to changes quickly. This is crucial for a large and technically challenging project where requirements may evolve as the project progresses.

The choice of a life cycle model depends on the project's specific needs, including the complexity of the software, the stability of the requirements, and the level of user involvement.

Is the number of loops of the spiral model fixed for different development projects? Justify

The number of loops in the Spiral Model is not fixed and varies depending on the specific needs of the development project. The Spiral Model is a risk-driven process model, which means that the development process is iterative and each loop through the spiral represents a phase of the project. Here's why the number of loops is not fixed:

- **Project Size and Complexity:** Larger and more complex projects may require more iterations to address all aspects of the software development life cycle and to manage risks effectively.
- **Risk Management:** The primary goal of the Spiral Model is to manage risks. Different projects have different risk profiles, and the number of loops will be influenced by the need to assess and mitigate these risks.

- **Customer Feedback:** The model allows for incorporating customer feedback and changing requirements. The number of iterations may increase if the feedback indicates significant changes to the project scope or direction.
- **Project Phases:** Each loop can represent a phase such as planning, risk analysis, engineering, and evaluation. The number of phases needed to complete the project successfully will determine the number of loops.

Therefore, the flexibility of the Spiral Model is one of its strengths, allowing it to adapt to the unique demands of each project. The model emphasizes iterative development, customer involvement, and risk management, making it suitable for projects where requirements are not well understood from the beginning or are expected to change over time.

With the help of a diagram explain spiral process model.

<https://www.geeksforgeeks.org/software-engineering-spiral-model/>

A group diary and time management system is intended to support the timetabling of meetings and appointments across a group of co-workers. When an appointment is to be made which involves several people, the system finds a common slot in each of their diaries and arranges the appointment for that time. If no common slots are available, it interacts with the user to rearrange their personal diary to make room for the appointment. Will you choose monolithic architecture or micro services architecture for this application? Justify

For the group diary and time management system you've described, I would choose a **microservices architecture**. Here's why:

- **Scalability:** Microservices allow each part of the system (e.g., appointment scheduling, diary management, user interaction) to scale independently as demand changes.
- **Flexibility:** With microservices, it's easier to implement and update the system's functionality. For example, if you want to change the way appointments are scheduled, you can update just that service without affecting the rest of the system.
- **Resilience:** If one microservice fails, it doesn't bring down the entire system. This is crucial for a system that needs to be available 24/7.

- **Technology Diversity:** Different microservices can use different technologies that are best suited for their specific tasks, which can lead to a more efficient system overall.
- **Ease of Deployment:** Microservices can be deployed independently, which is beneficial for continuous delivery and integration.

Given these advantages, a microservices architecture would be well-suited to handle the complex interactions and scalability requirements of a group diary and time management system.

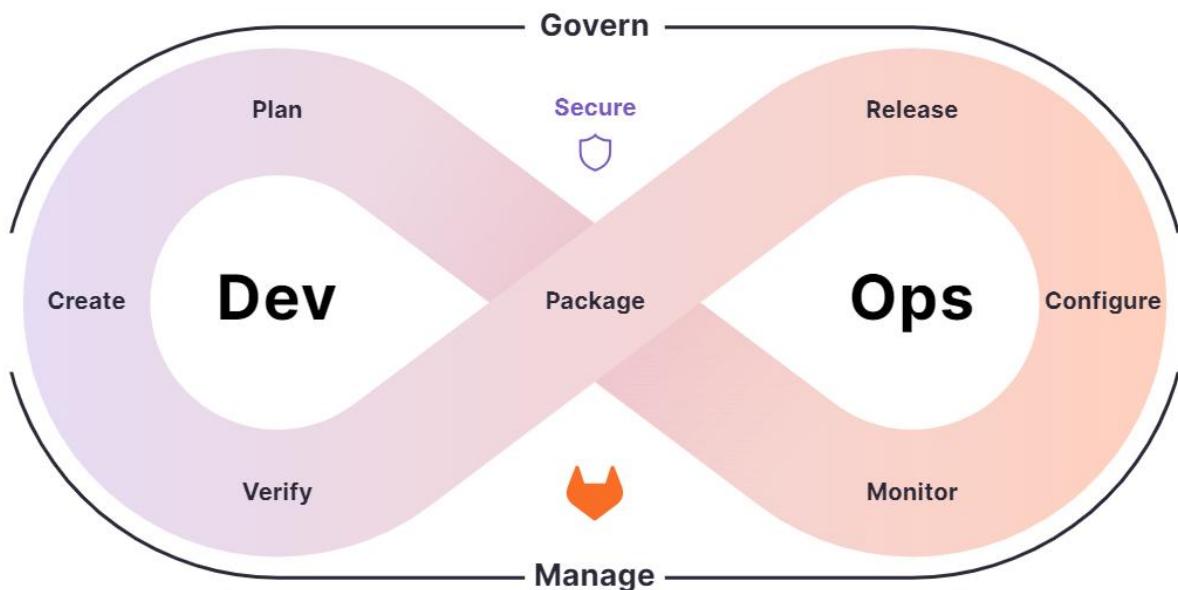
How can you apply DevOp for your project. Explain DevOp with the help of a diagram.

<https://about.gitlab.com/topics/devops/>

To apply DevOps to your project, you would follow a series of steps that integrate development (Dev) and IT operations (Ops) to improve collaboration and productivity by automating infrastructure, workflows, and continuously measuring application performance. Here's a step-by-step guide:

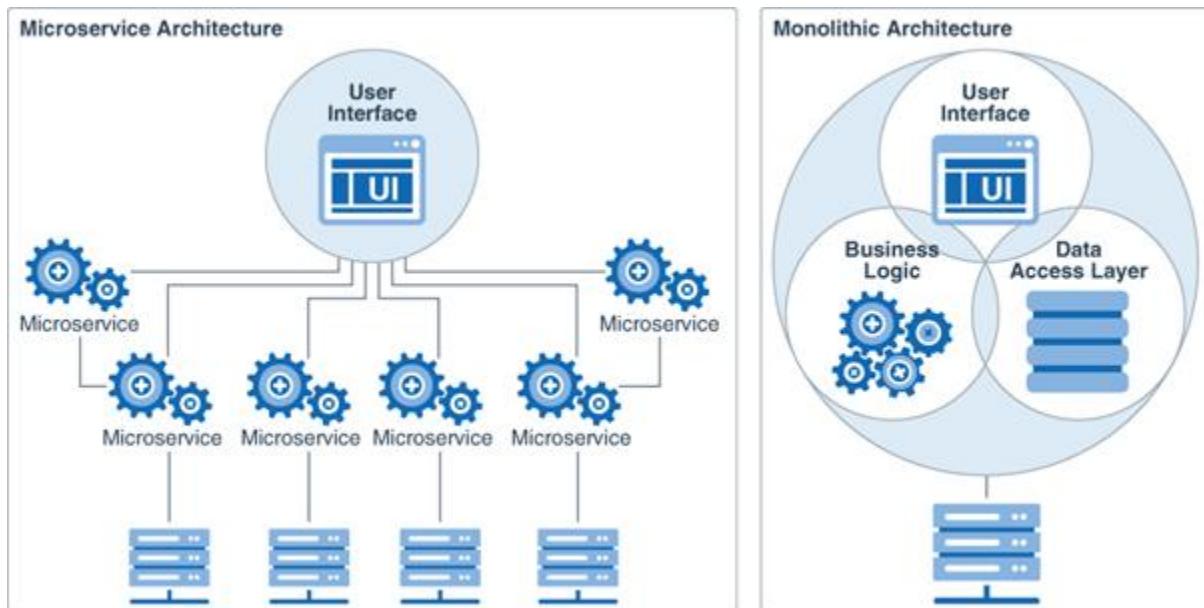
1. **Plan:** Define the scope, objectives, and timelines for your project. Use agile methodologies to manage tasks and sprints.
2. **Code:** Write code using version control systems to track changes and collaborate with other developers.
3. **Build:** Compile your code into executable files or packages. Automate this process to ensure consistency.
4. **Test:** Continuously test the code in an automated manner to catch bugs early.
5. **Release:** Automate the release process to move the code quickly and reliably into production.
6. **Deploy:** Automate deployment to ensure that the new code is safely deployed to the production environment without downtime.
7. **Operate:** Ensure the operational stability of the application with system monitoring and logging.
8. **Monitor:** Use monitoring tools to track the application's performance and other metrics in real-time.
9. **Feedback and Optimize:** Gather feedback from stakeholders and monitor data to continuously improve the application.

The DevOps process is cyclical, emphasizing ongoing improvement and efficiency. I'll try to create a diagram that represents the DevOps Process Model. Please note that the image will be shown directly to you in a separate frame after my response bubble.

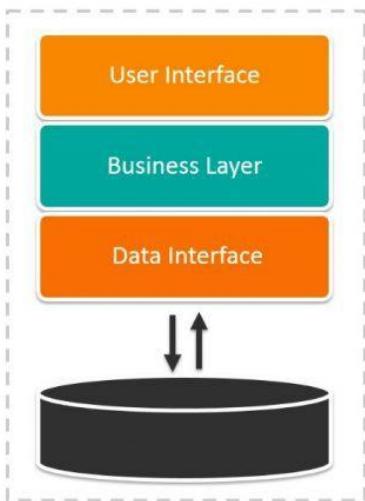


Describe Microservices architecture with the help of a diagram.

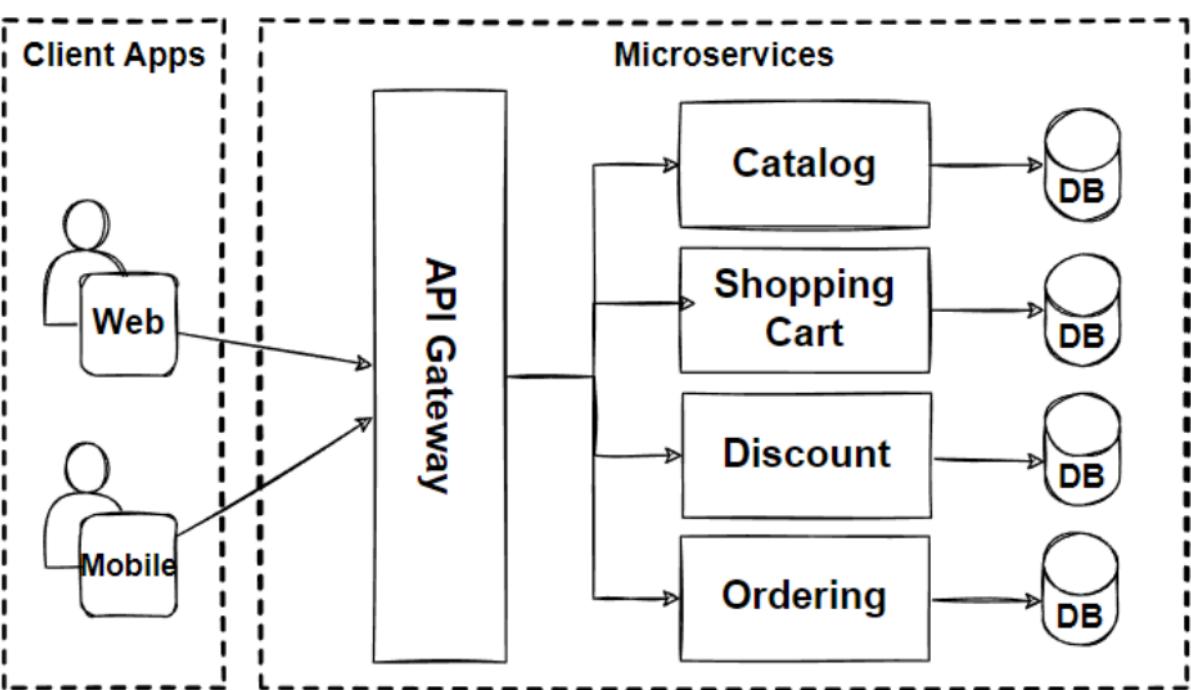
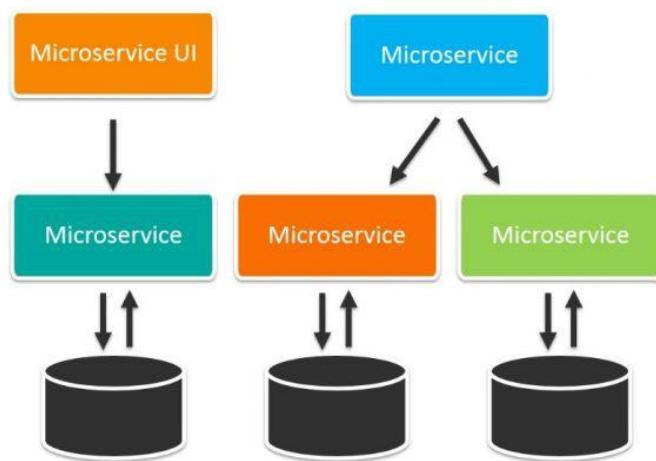
<https://www.geeksforgeeks.org/microservice-architecture-introduction-challenges-best-practices/>



Monolithic Architecture



Microservices Architecture



Microservices architecture is a method of developing software applications as a suite of small, independent services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. Each microservice focuses on a single business capability, is independently deployable, and can be developed by a small team.

Here's a high-level overview of a typical microservices architecture:

- **Client Layer:** This is where the user interfaces or external systems interact with the microservices through an API Gateway.

- **API Gateway:** Acts as the entry point for clients. It routes requests to the appropriate microservices and may also handle cross-cutting concerns like authentication and SSL termination.
- **Microservices:** Each microservice is a small, autonomous service that implements a single business capability within a bounded context. They can be developed, deployed, and scaled independently.
- **Data Layer:** Microservices often manage their own database or state, which ensures loose coupling and autonomy.
- **Infrastructure:** Common operational capabilities like service discovery, external configuration, or load balancing are often provided by the infrastructure.

The architecture promotes agility, scalability, and resilience, allowing teams to develop and deploy services independently and quickly.

For a detailed diagram and further reading, you can refer to the resources provided by [IBM](#), [Microsoft](#), and others¹². These resources offer in-depth explanations and diagrams that illustrate how microservices are structured and interact with each other.

Will exhaustive testing guarantee that the program is 100 percent correct? Justify

Exhaustive testing, which involves testing all possible inputs and preconditions, is **not feasible** for most software due to the sheer number of possible test cases. Even with a simple program, the number of potential paths through the code can be enormous. Here are some reasons why exhaustive testing does **not guarantee 100 percent correctness**:

- **Infinite Inputs:** Many programs can take an infinite number of different input values. It's impossible to test all possible inputs.
- **Combinatorial Explosion:** The number of possible paths through a program increases exponentially with the number of decisions in the code.
- **Changing Requirements:** Software requirements can change over time, which means that passing all tests at one point does not guarantee correctness later.
- **Environment and Interaction:** Programs do not run in isolation; they interact with operating systems, hardware, and other software, which can't all be exhaustively tested.
- **Human Error:** Test cases are designed by humans, who can make mistakes or overlook conditions.

Therefore, while thorough testing can significantly increase confidence in software correctness, it cannot prove it. Techniques like formal verification can prove certain properties of software, but these techniques are complex and not applicable to all kinds of software. The goal of testing is to find as many faults as possible within practical constraints of time and resources, not to prove absolute correctness.

What is Formal Technical Review (FTR) in SQA? Explain the steps to perform FTR.

Formal Technical Review (FTR) in Software Quality Assurance (SQA) is a structured process used to ensure that software products meet their specified requirements and are developed according to predefined standards. The steps to perform an FTR are as follows:

1. **Planning:** Determine the scope and objectives of the review. Select the parts of the software to be reviewed and identify the review team, which typically consists of 3 to 5 members.
2. **Preparation:** Distribute the necessary materials to the reviewers well in advance. Each reviewer should spend no more than 2 hours preparing for the FTR to maintain efficiency.
3. **Review Meeting:** Conduct the review meeting, which should last no longer than 2 hours. During the meeting, the team examines the software artifacts to uncover errors in logic, function, and implementation.
4. **Reporting:** Actively record all issues raised during the review. At the end of the meeting, consolidate these issues into a review issue list.
5. **Rework:** Based on the findings, the work product may be accepted without modification, rejected due to serious errors, or accepted provisionally with minor errors to be corrected.
6. **Follow-up:** Ensure that all agreed-upon modifications are made. A formal technical review summary report is then prepared, detailing what was reviewed, who reviewed it, and the findings and conclusions¹².

FTR is a key activity in SQA because it helps to identify defects early in the software development process, which can reduce the cost and time required for later debugging and rework. It also serves as a learning opportunity for junior engineers and promotes backup and continuity within the team¹.

A program reads 3 integer values. The 3 values are interpreted as representing the lengths of the sides of a triangle. The program checks if it is a triangle and prints a message that states whether the triangle is scalene, isosceles or equilateral and not triangle.

Create pseudo code for the function and calculate cyclomatic complexity using number of nodes and edges.

Here's the pseudo code for the function that determines the type of triangle based on the lengths of its sides:

```
FUNCTION CheckTriangle(a, b, c: INTEGER): STRING
    IF a + b > c AND a + c > b AND b + c > a THEN
        IF a = b AND b = c THEN
            RETURN "Equilateral Triangle"
        ELSE IF a = b OR a = c OR b = c THEN
            RETURN "Isosceles Triangle"
        ELSE
            RETURN "Scalene Triangle"
        END IF
    ELSE
        RETURN "Not a Triangle"
    END IF
END FUNCTION
```

To calculate the cyclomatic complexity using the number of nodes and edges, we can construct a control flow graph (CFG) of the above pseudo code and use the formula:

$$\text{Cyclomatic Complexity} = E - N + 2P$$

Where:

- E = Number of edges
- N = Number of nodes
- P = Number of connected components (For a single program, P = 1)

For the given pseudo code, let's identify the nodes and edges:

- Nodes include the start of the function, each condition check, and the end of the function (return statements).
- Edges include the flow from one node to another.

Assuming the CFG for the pseudo code has:

- 8 nodes (Start, 3 condition checks, 4 return statements)
- 9 edges (The flow between the nodes)

And since we have only one connected component ($P = 1$), the cyclomatic complexity would be:

$$\text{Cyclomatic Complexity} = 9 - 8 + 2(1) = 3$$

Therefore, the cyclomatic complexity of the function is 3. This means there are three linearly independent paths through the program's source code.

**What are the different types of risks that a typical software project might suffer from? Give examples of each type of risk.
What are the strategies for risk mitigation?**

<https://www.geeksforgeeks.org/different-types-of-risks-in-software-project-development/>

In a typical software project, there are several types of risks that can arise. Here are some of the common types of risks along with examples and strategies for risk mitigation:

1. Schedule Risk

- *Example:* A project may fall behind schedule due to underestimated task durations or unforeseen technical difficulties.
- *Mitigation:* Implement agile methodologies, break down tasks into smaller, manageable units, and conduct regular progress reviews.

2. Budget Risk

- *Example:* The project may exceed its budget due to scope creep or unexpected increases in resource costs.
- *Mitigation:* Careful budget planning, regular monitoring of expenditures, and maintaining a contingency fund for unexpected costs.

3. Operational Risk

- *Example:* Day-to-day operational challenges such as insufficient resources or conflicts between tasks and employees.
- *Mitigation:* Effective resource management, clear communication channels, and robust process documentation.

4. Technical Risk

- *Example:* Technical issues such as integration problems, software bugs, or performance bottlenecks.
- *Mitigation:* Conduct thorough testing, use continuous integration and deployment practices, and keep technical documentation up to date.

5. Business Risk

- *Example:* The risk of developing a product that does not meet market needs or losing key personnel.
- *Mitigation:* Market analysis, prototyping, user testing, and having a well-defined product vision.

6. Project Risk

- *Example:* Risks related to project management, such as lack of clear objectives or poor stakeholder engagement.
- *Mitigation:* Establish clear project goals, engage stakeholders regularly, and use project management tools.

7. Quality Risk

- *Example:* The final product may not meet quality standards or customer expectations.
- *Mitigation:* Implement quality assurance processes, conduct code reviews, and involve customers in the testing phase.

8. Security Risk

- *Example:* Potential vulnerabilities in the software that could be exploited by malicious actors.
- *Mitigation:* Regular security audits, adherence to security best practices, and staying updated with the latest security patches.

Each type of risk requires a specific mitigation strategy, often involving a combination of planning, monitoring, and proactive measures to address potential issues before they become problematic¹²³⁴. It's important to identify and assess these risks early in the project lifecycle to implement effective mitigation strategies. Regular risk assessments and reviews should be part of the project management process to ensure that risks are managed throughout the project³⁴.

Do a functional decomposition/Work breakdown structure of the on-line social networking site. Estimate the size of each function in LOC. Assuming that a organization produces 450 LOC/pm with a burdened labor rate of Rs.7000 per person-month, estimate the effort and cost required to build the software using LOC-based estimation technique.(Assume suitable data for functional decomposition)

To perform a functional decomposition and work breakdown structure (WBS) for an online social networking site, we would break down the project into smaller, more manageable components or functions. Here's an example of how this might look:

- 1. User Management**
 - Registration
 - Login/Logout
 - Profile Management
- 2. Social Interaction**
 - Friend Requests
 - Messaging
 - Notifications
- 3. Content Management**
 - Posting Updates
 - Comments
 - Likes and Reactions
- 4. Search Functionality**
 - Search Users
 - Search Content
- 5. Data Management**
 - Database Design
 - Data Security
 - Backup and Recovery
- 6. Infrastructure**
 - Server Setup
 - Network Configuration
 - Maintenance

For the sake of estimation, let's assume the following sizes in Lines of Code (LOC) for each function:

1. User Management: 10,000 LOC
2. Social Interaction: 15,000 LOC
3. Content Management: 8,000 LOC
4. Search Functionality: 6,000 LOC
5. Data Management: 12,000 LOC
6. Infrastructure: 5,000 LOC

Total estimated size: 56,000 LOC

Given that the organization produces 450 LOC per person-month (pm), we can estimate the effort required:

Effort = Total LOC / Productivity Effort = 56,000 LOC / 450 LOC/pm Effort ≈ 124.44 pm

The cost can be estimated using the burdened labor rate:

Cost = Effort * Burdened Labor Rate Cost = 124.44 pm * Rs.7000/pm Cost ≈ Rs.870,800

Therefore, the estimated effort to build the software is approximately 124.44 person-months, and the estimated cost is approximately Rs.870,800. Please note that these figures are based on the assumed data for functional decomposition and should be adjusted according to the actual project details.

What are the different types of Black box testing methods? Give examples of each.

Black box testing is a method where the tester does not have knowledge of the internal workings of the application. Instead, the tester focuses on the input and output of the software system. Here are some of the different types of black box testing methods:

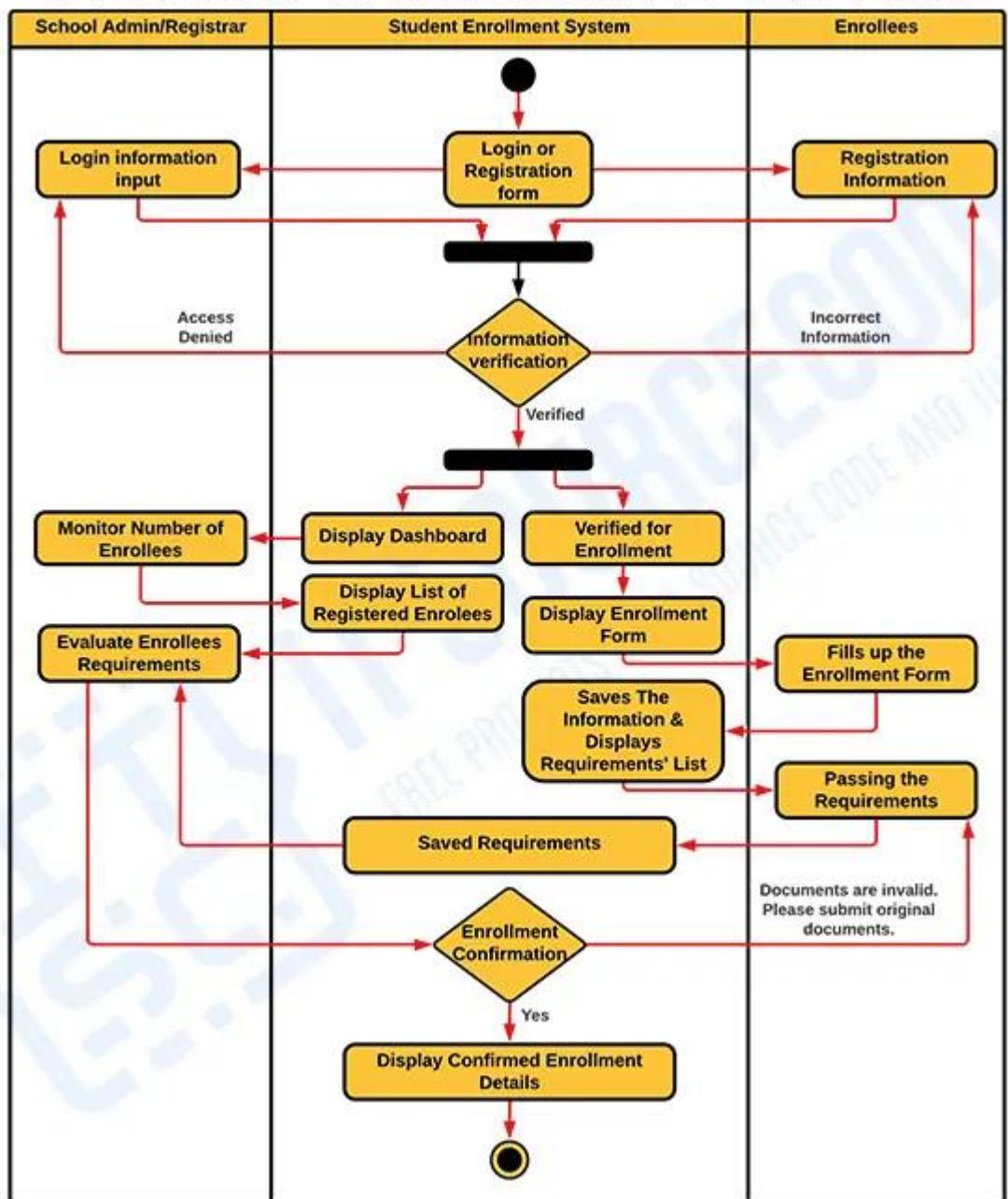
1. **Equivalence Partitioning:** This technique divides input data into equivalent partitions which can be used to generate test cases. It reduces the total number of test cases to a finite set of testable classes.
 - o *Example:* If an input accepts a range of values from 1 to 500, the test cases can be derived for values like 1, 250, and 500.
2. **Boundary Value Analysis:** This method involves testing at the boundaries between partitions. It is often used in conjunction with equivalence partitioning.
 - o *Example:* For the same range of 1 to 500, test cases would include values like 0, 1, 499, and 500.
3. **Decision Table Testing:** This is used when dealing with a combination of inputs that produce different results. It involves creating a table to represent combinations of inputs that can produce different outputs.
 - o *Example:* If a discount rate applies to a purchase over \$100 and a different rate for purchases over \$500, a decision table would help to understand the different outcomes.
4. **State Transition Testing:** This method is used where the system transitions from one state to another based on events or conditions.
 - o *Example:* In a login process, the system might transition from 'waiting for input' to 'user authenticated' state after the correct credentials are entered.

5. **Use Case Testing:** This technique uses use cases to identify test cases that cover the entire system.
 - *Example:* A use case might describe a user ordering a product online, and the test cases would cover each step in this process.
6. **Error Guessing:** This is a technique where the tester uses experience to guess the problematic areas of the application.
 - *Example:* A tester might guess that a field accepting date input might fail for leap years and would create a test case for February 29th.

These methods help ensure that the software behaves as expected in all scenarios that a user might encounter¹²³⁴. The goal is to find errors and defects by examining the most critical or vulnerable parts of the system.

What is swimlanes? Draw Activity Diagram with swimlanes for student admission process in Engineering College.

STUDENT ENROLLMENT SYSTEM



ACTIVITY DIAGRAM

<https://itsourcecode.com/uml/activity-diagram-for-student-enrollment-system-uml/>

<https://www.geeksforgeeks.org/swim-lanes-in-activity-diagram/>

Swimlanes are a visual element used in process flow diagrams or flowcharts that separate activities into different horizontal or vertical “lanes.” Each lane represents a different department, group, or individual responsible for certain tasks within the system.

overall process. This helps to clarify who is responsible for each step in a process and can be particularly useful in complex processes involving multiple stakeholders.

List four advantages of using design patterns? Explain Adaptor and singleton design patterns.

[Design Patterns: Understand The Importance With Real Life Examples - GeeksforGeeks](#)



Design patterns offer several advantages in software development:

1. **Reusability:** Design patterns provide proven solutions to common problems, which can be reused across different projects. [This saves time and effort in the development process¹.](#)
2. **Clarity and Communication:** [They serve as a common language among developers, making it easier to communicate complex design concepts and solutions¹.](#)
3. **Best Practices:** [Design patterns encapsulate best practices that have been evolved and refined over time by experienced developers².](#)
4. **Scalability and Maintainability:** [By promoting the use of clean and organized code structures, design patterns enhance the scalability and maintainability of software applications³.](#)
5. **Performance:** Some design patterns, when implemented correctly, can contribute to improved performance. For example, patterns like the Flyweight pattern can help in efficient memory usage.

Adapter Design Pattern: The Adapter pattern allows objects with incompatible interfaces to collaborate. It acts as a bridge between two different interfaces, making them compatible without changing their existing code. [For example, if you have a Bird class with a makeSound method and a ToyDuck class with a squeak method, and you want to make the Bird object compatible with the ToyDuck interface, you would use an adapter to translate the calls appropriately⁴.](#)

Singleton Design Pattern: The Singleton pattern ensures that a class has **only one instance** and provides a **global point of access** to it. This is useful when exactly one object is needed to coordinate actions across the system, such as a database connection or a configuration manager. The Singleton pattern can be implemented in various ways to ensure that there is only one instance of a class, even in a multithreaded environment⁵.

These patterns are fundamental tools for building reliable and efficient software systems.

Develop the SRS document for the First year Engineering Admission System.

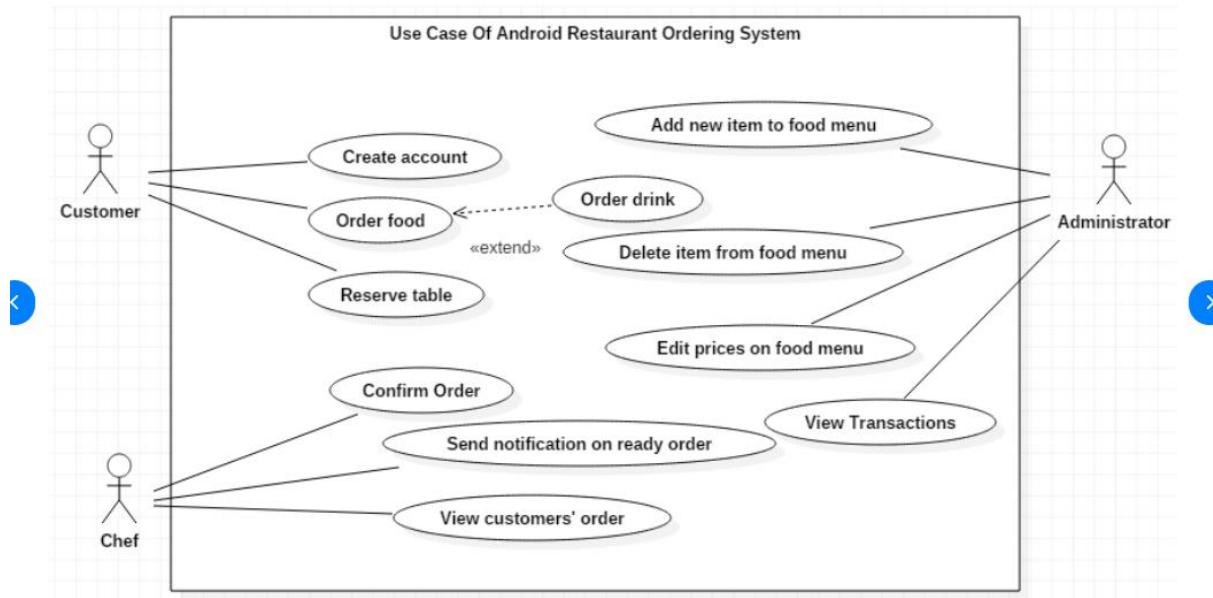
<https://www.scribd.com/document/310631366/SRS-of-Online-Admission-System>

Aug 2023

Consider that an Order consists of one or more Items. Each Item contains the name of the item, its quantity and the date by which it is required. Each order item is described by an Item Specification object having details such as its vendor addresses, unit price, and manufacturer. What is the relationship that exists between Item and Item Specification classes? Is it Dependency/ Aggregation/ Generalization/Composition? Justify.

The relationship between the **Item** and **Item Specification** classes is one of **Composition**. In object-oriented design, composition is a strong form of association where the composed object cannot exist independently of the composite. Here, each **Item** is associated with an **Item Specification**, which contains details like vendor addresses, unit price, and manufacturer that are essential to the **Item**. The **Item Specification** is part of the **Item**, and if the **Item** were to be removed or deleted, the associated **Item Specification** would also cease to exist. This is indicative of a whole-part relationship, where **Item Specification** is a part of **Item**, making it a composition rather than just an aggregation or generalization. Dependency is a weaker form of association and does not fit the described relationship where **Item** contains **Item Specification** details as an integral part of its definition.

Identify uses cases for an order processing system and draw use case diagram.



<https://online.visual-paradigm.com/diagrams/templates/use-case-diagram/uml-use-case-diagram-order-process-system/>

<https://www.freeprojectz.com/use-case/order-processing-system-use-case-diagram>

Certainly! Here are some typical use cases for an order processing system:

1. **Place Order:** A customer selects items and places an order.
2. **Process Payment:** The system processes the customer's payment.
3. **Check Inventory:** The system checks the inventory for item availability.
4. **Update Inventory:** The inventory is updated after an order is placed.
5. **Generate Invoice:** An invoice is generated for the order.
6. **Ship Order:** The order is packed and shipped to the customer.
7. **Track Order:** The customer can track the status of their order.
8. **Return/Exchange Items:** Handling returns or exchanges requested by the customer.
9. **Manage Customer Information:** Update and maintain customer profiles.
10. **Handle Customer Queries:** Respond to customer inquiries about orders.

For the use case diagram, I'll try to create that. Please note that the diagram will be a simplified representation and may not cover all the complexities of a real-world system.

Consider a smart home system capable of controlling various functionality such as switching on light and adjusting the AC temperature inside the room. The system employs different sensors to understand and adjust to room conditions and react accordingly. The system has photoresistors to detect the light condition of the room to switch on the lights and a thermostat to detect the temperature changes to control the AC.

Identify 2 IS-A relationship and 2 HAS-A relationship in given scenario. Draw class diagram of the system.

In the scenario of a smart home system, here are two IS-A relationships and two HAS-A relationships:

IS-A Relationships:

1. **LightController IS-A Controller**: The LightController is a type of Controller that specifically manages the lighting system.
2. **Thermostat IS-A Sensor**: The Thermostat is a type of Sensor that measures temperature.

HAS-A Relationships:

1. **SmartHomeSystem HAS-A LightController**: The SmartHomeSystem contains a LightController to manage the lights.
2. **SmartHomeSystem HAS-A Thermostat**: The SmartHomeSystem contains a Thermostat to control the AC temperature.

I'll try to create a class diagram that represents these relationships and the overall structure of the smart home system. Please note that the image will be shown directly to you in a separate frame after my response bubble.

Identify 2 reasons for need of a software life cycle model?

The need for a software life cycle model arises primarily for two reasons:

1. **Structure and Framework**: A software life cycle model provides a structured framework that helps to plan, organize, and control the process of developing an information system. This structure ensures that the complex process of software development is manageable and not chaotic.
2. **Quality and Reliability**: It ensures that the software is developed in a systematic and disciplined manner which leads to the creation of a quality product. The life cycle model facilitates thorough testing, documentation,

and **understanding** of the software development process, which contributes to the reliability and quality of the final product.

These models guide the development process, making it easier to track progress and make adjustments as needed. They also help in risk management and resource allocation throughout the different phases of software development.

A mining company named Galax\ Mining Company Ltd. (CMC) has mines located at various places in India. It has about fifty different mine sites spread across eight states. The company employs a large number of miners at each mine site. Mining being a risky profession, the company intends to operate a special provident fund, which would exist in addition to the standard provident fund that the miners already enjoy. The main objective of having the special provident fund (SPF) would be quickly distribute some compensation before the standard provident amount is paid. According to this scheme, each mine site would deduct SPF installments from each miner every month and deposit the same with the CSPFC (Central Special Provident Fund Commissioner). The CSPFC will maintain a 11 details regarding the SPF amounts collected from the miners. CMC employed a reputed software vendor Adventure Software Inc. to undertake the task of developing the software for automating the maintenance of SPF records of all employees. CMC realized that besides saving manpower on book keeping work, the software would help in speedy settlement of claim cases. CMC indicated that the amount it can afford for this software to be developed and installed is Rs. 1 million. Adventure Software Inc. deputed their project manager to create the system. Elaborate the project phases and tasks in each phase assuming the company followed Incremental life cycle model.

In the Incremental life cycle model, the project is divided into increments, each of which delivers a portion of the functionality. The project phases and tasks for the SPF software development for Galax Mining Company Ltd. (CMC) would typically include:

1. Requirements Analysis and Project Planning:

- Gather detailed requirements for the SPF system.
- Define the scope and objectives of the project.

- Develop a project plan, including timelines and resource allocation.
 - Establish a budget within the Rs. 1 million affordability.
- 2. System Design:**
- Create the overall system architecture.
 - Design the database schema for maintaining SPF records.
 - Define the user interface and experience (UI/UX) for the software.
- 3. Incremental Development:**
- Develop the first increment with basic functionality such as miner registration and SPF installment tracking.
 - Test the first increment and gather feedback from CMC.
 - Plan subsequent increments based on priority features and feedback.
- 4. Testing and Integration:**
- Perform unit testing, integration testing, and system testing for each increment.
 - Ensure that each increment integrates smoothly with the existing system components.
- 5. Deployment:**
- Deploy each increment in a live environment after testing.
 - Train CMC staff on how to use the new system features.
- 6. Maintenance and Evaluation:**
- Provide ongoing support and maintenance for the software.
 - Evaluate the system performance and make necessary adjustments.
- 7. Feedback and Iteration:**
- Collect feedback from CMC and the miners on the system's usability and effectiveness.
 - Plan for the next increment incorporating the feedback received.
- 8. Final Delivery:**
- Once all increments have been developed, tested, and deployed, ensure that the complete system meets CMC's requirements.
 - Deliver the final product and close the project.

By following the Incremental life cycle model, Adventure Software Inc. can deliver the SPF system in manageable pieces, allowing CMC to start using the system early and provide feedback that can be incorporated into later increments. This approach also helps in managing risks and ensuring that the project stays within the allocated budget.

Illustrate with an example, balancing of DFD.

<https://www.geeksforgeeks.org/developing-dfd-model-of-system/>

[Balancing DFD | slideum.com](https://slideum.com/presentation/123456789/balancing-dfd)

https://www.youtube.com/watch?v=qyXtZvX3Lok&ab_channel=PUPTBSIT

<https://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc38088.1610/doc/html/rad1232026236784.html>

A supermarket needs to develop the following software to encourage

regular customers. For this, the customer needs to supply his/her residence address, telephone number, and the driving license number.

Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. A customer can present his

(N to the check out staff when he makes any purchase. In this case,

the value of his purchase is credited against his CN. At the end of each

year, the supermarket intends to award surprise gifts to 10 customers

who make the highest total purchase over the year. Also, it intends to

award a 22 caret gold coin to every customer whose purchase

exceeded Rs. 10,000. The entries against the CN are reset on the last

day of every year after the prize winners' lists are generated.

Draw level 0, I, 2 DFD for the scenario. (Level 2 for one DFD process)

Draw this

Barry is assigned to a project team to develop a store management system for a chain of sandwich shops. The system must have features to take order, make sandwich, collect payment, hand over order, generate day wise order summary, check inventory etc. Identify the EI, EO, EQ, ILF etc in the system, calculate function point and estimate the effort required for the system.

Weighting Factor for computing FP

Parameter	Simple	Average	Complex
External Input (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiry (EQ)	3	4	6
Internal Logical File (ILF)	7	10	15
External Interface File (EIF)	5	7	10

Give 2 disadvantages of Function Point estimation

Certainly! Here are two disadvantages of Function Point Estimation:

1. **Subjectivity**: Function Point Estimation requires subjective evaluations, which involve a significant amount of judgment. [This can lead to inconsistencies and inaccuracies in the estimation process¹.](#)
2. **Conversion to LOC**: Many effort and cost models are based on Lines of Code (LOC), so function points often need to be converted to LOC for compatibility with these models. [This conversion can introduce additional complexity and potential for error¹.](#)

These disadvantages highlight some of the challenges in applying Function Point Estimation to software project management.

Suggest a mitigation plan for the risk given below:

Software requires interface with other technologies on which the project team has no experience.

EXPLAIN 3 steps in software risk management.

To mitigate the risk of interfacing with technologies that the project team has no experience with, a mitigation plan could include the following steps:

1. **Training and Knowledge Acquisition:** Invest in training for the project team to learn the new technologies. This could involve formal courses, workshops, or self-study. Additionally, hiring or consulting with experts who have experience with these technologies can provide valuable insights and accelerate the learning process.
2. **Prototyping:** Before full-scale development, create prototypes that involve the new interfaces. This allows the team to gain practical experience and identify potential issues early on.
3. **Incremental Integration:** Integrate the new technologies incrementally rather than all at once. Start with non-critical areas to reduce the impact of any problems that arise and gradually move to more critical components as the team's expertise grows.

Regarding the three steps in software risk management, they are typically:

1. **Risk Identification:** Recognize potential risks that could affect the project, including technical, project, and business risks.
2. **Risk Analysis:** Evaluate the likelihood and impact of each identified risk. This helps in prioritizing which risks need to be addressed first.
3. **Risk Mitigation:** Develop strategies to either prevent the risks from occurring or reduce their impact if they do occur. This includes creating contingency plans for how to handle risks if they become actual issues¹²³⁴.

These steps help ensure that risks are managed proactively, reducing the likelihood of project delays, cost overruns, and other issues related to unforeseen challenges. Implementing a structured risk management process is crucial for the successful completion of software projects.

State the need for prototyping and" how this approach save cost , effort and resources. Elaborate the steps followed in Prototype life cycle

model.

Prototyping is an essential approach in software development for several reasons:

- **Early Visualization:** It allows stakeholders to see and interact with a working model of the product early in the development process.
- **Feedback Incorporation:** Stakeholders can provide immediate feedback, which can be quickly incorporated into the design before costly development efforts are undertaken.

- **Requirement Validation:** Prototyping helps in validating requirements and can reveal misunderstandings or missing functionality that might not have been evident in a static specification.
- **Cost and Effort Reduction:** By identifying issues early, prototyping can prevent the need for extensive revisions later in the project, saving both time and money.
- **Improved User Satisfaction:** It ensures that the final product is more likely to meet user needs and expectations, leading to higher satisfaction and reduced need for future modifications.

The steps in the Prototype life cycle model are:

1. **Identify Basic Requirements:** Determine the basic requirements for the system, focusing on the user interface and user interaction with the system.
2. **Develop Initial Prototype:** Create an initial prototype that includes only user interfaces.
3. **User Evaluation:** Have users evaluate the prototype, providing feedback on its functionality and usability.
4. **Revise and Enhance the Prototype:** Based on user feedback, revise the prototype to better meet the needs and requirements.
5. **Refine Prototype:** Continue to refine the prototype through multiple iterations, improving the design and functionality with each cycle.
6. **Finalize System:** Once the prototype meets all the requirements and user expectations, finalize the design for production.

By following these steps, the Prototype life cycle model helps in creating a more user-centric product while optimizing resources and minimizing risks associated with software development. Prototyping is a proactive measure that can lead to significant cost savings and a more successful product launch.

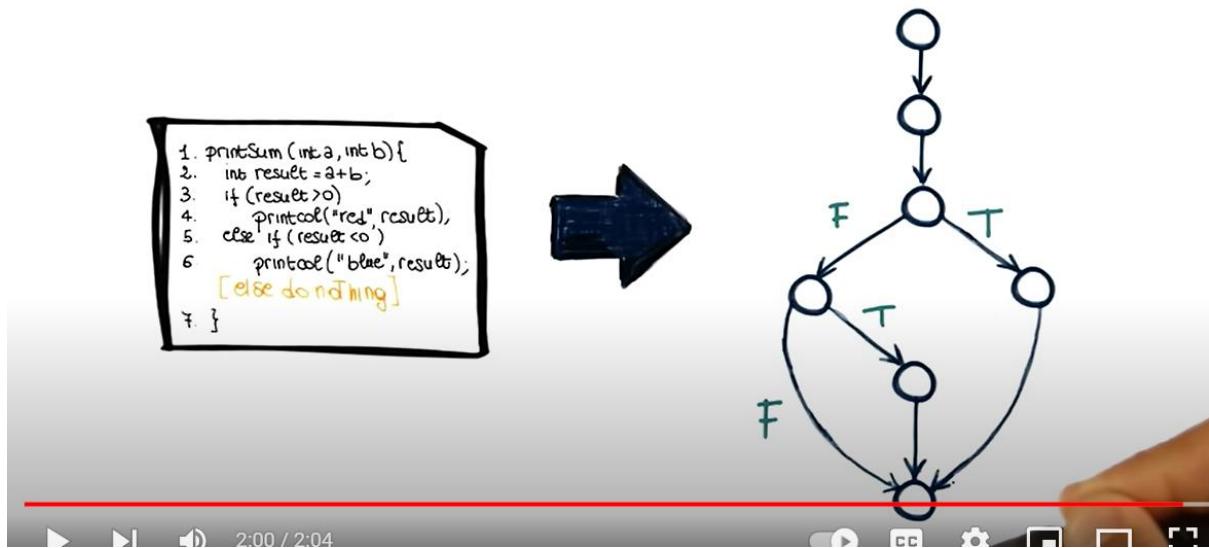
Explain the metrics for the design Model

<https://www.geeksforgeeks.org/metrics-for-the-design-model-of-the-product/>

Consider the following function. Draw Control flow graph. From the control flow graph, determine its Cyclomatic complexity and Find linearly independent paths. Design a test cases for the function.

```
int find-maximum(int i, int j, int k)
{
    int max;
    if (i>j) then
        if(i>k) then max = i;
        else max = k;
    else if (j>k) max = j;
    else max = k;
    return (max);
}
```

DIGRESSION : CONTROL FLOW GRAPHS



<https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/>

For Cyclomatic Complexity calculations

<https://www.gatevidyalay.com/tag/control-flow-graph-cyclomatic-complexity/>

How can you apply DevOps for a project. Illustrate with example the different phases of DevOps.

To apply DevOps to a project, you would integrate development (Dev) and IT operations (Ops) to streamline the software development lifecycle. Here's how you can apply DevOps with an example of the different phases:

1. **Continuous Development:** This phase involves planning and coding. For example, using version control systems like Git to manage code changes and collaborate.
2. **Continuous Integration:** In this phase, code changes are frequently integrated into the main branch, ensuring that new code does not break the build. Tools like Jenkins can be used for automated builds and testing.
3. **Continuous Testing:** Automated tests are run to validate the code changes. Tools like Selenium can be used to automate browser testing.
4. **Continuous Deployment:** This phase involves the automated deployment of code to production environments. Tools like Docker can be used to containerize applications for consistent deployment.
5. **Continuous Monitoring:** Monitoring the application's performance to identify issues. Tools like Nagios can be used to monitor system health and performance.
6. **Continuous Feedback:** Gathering feedback from users to inform future development cycles. This can be done through user analytics and bug reporting tools.
7. **Continuous Operations:** Automating the release process to ensure that the application is always up-to-date with minimal downtime.

For example, a company developing a web application might use DevOps to automate the setup of their development, testing, and production environments using infrastructure as code tools like Terraform. They would write code and commit changes to a Git repository, which triggers a Jenkins pipeline to build and test the application. Successful builds are then deployed to a Docker container in a staging environment for further testing. Once approved, the changes are deployed to production. [The application is monitored using Nagios, and any issues are reported back to the development team for resolution in the next iteration¹².](#)

By following these phases, DevOps helps in achieving faster delivery times, higher quality software, and more efficient and collaborative development processes.

What are the pros and cons of Design Patterns. Illustrate Singleton Pattern with example

Design patterns are a set of best practices and solutions to common design problems in software engineering. Here are some pros and cons of using design patterns:

Pros:

- **Reusability:** Design patterns provide a standard solution that can be reused across different projects.
- **Best Practices:** They encapsulate the experience and best practices of software development.
- **Readability:** Using design patterns can make the code more readable and easier to understand for other developers.
- **Maintainability:** They help in creating a maintainable codebase by providing proven development paradigms.

Cons:

- **Complexity:** Sometimes, design patterns can add unnecessary complexity to the code if not used appropriately.
- **Overuse:** Overusing design patterns can lead to a situation where the code becomes more difficult to understand and maintain.
- **Learning Curve:** There is a learning curve associated with understanding and implementing design patterns correctly.

Singleton Pattern Example: The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. Here's an example in Java:

```
public class Singleton {  
    // The private static variable of the same class that is the only  
    // instance of the class.  
    private static Singleton singleInstance = null;  
  
    // Private constructor to restrict instantiation of the class from  
    // other classes.  
    private Singleton() {}  
  
    // Public static method that returns the instance of the class, this is  
    // the global access point for outer classes to get the instance of the  
    // singleton class.  
    public static Singleton getInstance() {  
        if (singleInstance == null) {  
            singleInstance = new Singleton();  
        }  
        return singleInstance;  
    }  
  
    // Any other methods  
}
```

In this example, the `Singleton` class has a private constructor and a private static variable that holds the single instance of the class. The `getInstance()` method

ensures that only one instance of the class is created and provides a global point of access to that instance. This pattern is useful when exactly one object is needed to coordinate actions across the system.

May 2021

Match the following

- | | |
|--------------------------|--------------------------------|
| a) Requirement analysis | 1) Meet user needs |
| b) Agile spikes | 2) Matches SRS |
| c) Unit Testing | 3) Potential solution |
| d) Software Verification | 4) Implementation phase |
| e) Software Validation | 5) Non-functional requirements |

Here's the matching for the given items:

- a) Requirement analysis - **5) Non-functional requirements** Requirement analysis involves understanding both functional and non-functional requirements of the system.
- b) Agile spikes - **3) Potential solution** Agile spikes are used to explore potential solutions to complex problems in the Agile development process.
- c) Unit Testing - **4) Implementation phase** Unit testing is conducted during the implementation phase to ensure that individual units of code work as intended.
- d) Software Verification - **2) Matches SRS** Software verification is the process of checking that the software matches the specified requirements (SRS - Software Requirement Specification).
- e) Software Validation - **1) Meet user needs** Software validation ensures that the software meets the user's needs and that it fulfills its intended use.

How is agility achieved in agile software development model?

Agility in the agile software development model is achieved through several key practices and principles that emphasize flexibility, collaboration, and iterative progress. Here are some ways agility is achieved:

- **Iterative Development:** Agile development is carried out in short cycles or iterations, allowing for frequent reassessment and adaptation of plans.
- **Customer Collaboration:** Working closely with customers to understand their needs and making them a part of the development process ensures that the product evolves as per user requirements.

- **Cross-functional Teams:** Agile teams are usually cross-functional, meaning they have all the skills necessary to deliver a working increment of software.
- **Adaptive Planning:** Agile methodologies promote flexible planning that can adapt to changes rather than following a fixed path.
- **Continuous Feedback:** Regular feedback from users and stakeholders is incorporated into the development process, leading to continuous improvement.
- **Simplicity:** Agile methods encourage simplicity in design and process, which contributes to faster development and easier maintenance.
- **Sustainable Pace:** Agile promotes a sustainable work pace, avoiding burnout and maintaining high productivity over time.

These practices help agile teams to respond to changes quickly and efficiently, delivering value to customers early and continuously throughout the project.

How is progress measure achieved in agile development?

In Agile development, progress is measured through various metrics that provide insight into productivity, predictability, quality, and value throughout the software development lifecycle. Here are some key metrics used to measure progress in Agile development:

- **Velocity:** This measures the amount of work a team can complete in a single sprint and is used to plan future sprints¹.
- **Burndown Charts:** These show the amount of work remaining in a sprint or project, helping teams understand if they are on track to complete their work in the time available¹.
- **Burnup Charts:** Similar to burndown charts, but they show the amount of work completed over time as well as the total amount of work, providing a broader view of progress¹.
- **Cumulative Flow Diagrams (CFD):** These provide a visual representation of the various stages of work in a project, from “To Do” to “Done,” helping identify bottlenecks².
- **Lead Time and Cycle Time:** Lead time measures the time from customer request to product delivery, while cycle time measures the time it takes to complete work items once they start².
- **Work In Progress (WIP):** This tracks the number of tasks that are currently in progress, ensuring that teams are not overburdened and can focus on completing tasks².

These metrics help Agile teams set benchmarks, measure against goals, evaluate performance, and continuously improve their processes. It's important to choose metrics that align with the team's goals and that are clearly defined and understood.

by all team members. [The right combination of metrics can provide a comprehensive view of Agile performance, covering aspects such as predictability, productivity, quality, and value².](#)

Software architecture is important to meet software quality and reusability. Discuss with example.

Software architecture plays a crucial role in achieving software quality and reusability. It serves as the blueprint for both the system and the project, providing the necessary structure to ensure that the software meets quality requirements and can be reused in other projects.

Quality: A well-designed architecture can significantly improve the quality of software by ensuring that it is scalable, reliable, maintainable, and performs well. For example, a microservices architecture allows for independent deployment of different services, which can improve scalability and reliability. If one service fails, it doesn't necessarily bring down the entire system, thus enhancing fault tolerance.

Reusability: Software architecture that emphasizes modularity and loose coupling makes it easier to reuse components in other applications. For instance, if an application is built using a layered architecture, individual layers or components can be reused across different systems that require similar functionality. This not only saves development time but also ensures consistency and reduces the potential for errors.

By carefully considering software architecture, developers can create systems that not only meet the current requirements but are also adaptable to future changes and can serve as a foundation for new projects. This strategic approach to software development can lead to significant cost savings and improved software longevity.

Project schedule becomes a road map that defines the tasks and milestones

to be tracked and controlled.

List 5 different ways that can be used to perform tracking.

Project tracking is essential for ensuring that a project stays on schedule and within budget. Here are five different ways to perform tracking:

1. **Gantt Charts:** These provide a visual timeline for the project, showing when each task should be started and completed. They help in tracking progress against planned schedules.
2. **Critical Path Analysis:** This method identifies the sequence of crucial and interdependent steps that determine the project's duration. Tracking the critical path helps in identifying which tasks need to be prioritized to stay on schedule.
3. **Earned Value Management (EVM):** EVM is a technique that assesses project performance and progress in an objective manner by comparing the amount of work planned with what has actually been completed, and the costs incurred to accomplish it.
4. **Kanban Boards:** Kanban boards are visual tools that track the status of work items in a project. They help teams visualize workflow and identify bottlenecks in real-time.
5. **Scrum Meetings (Stand-ups):** Daily scrum meetings are short meetings used in Agile project management to ensure team members are aware of each other's progress and challenges. They provide an opportunity to adjust tasks and priorities on a daily basis.

Each of these methods offers a different approach to tracking and can be used individually or in combination to provide a comprehensive view of project progress.

Software architecture is important to meet software quality and reusability. Discuss with example.

Software architecture plays a crucial role in achieving software quality and reusability. It serves as the blueprint for both the system and the project, providing the necessary structure to ensure that the software meets quality requirements and can be reused in other projects.

Quality: A well-designed architecture can significantly improve the quality of software by ensuring that it is scalable, reliable, maintainable, and performs well. For example, a microservices architecture allows for independent deployment of different services, which can improve scalability and reliability. If one service fails, it doesn't necessarily bring down the entire system, thus enhancing fault tolerance.

Reusability: Software architecture that emphasizes modularity and loose coupling makes it easier to reuse components in other applications. For instance, if an application is built using a layered architecture, individual layers or components can be reused across different systems that require similar functionality. This not only saves development time but also ensures consistency and reduces the potential for errors.

By carefully considering software architecture, developers can create systems that not only meet the current requirements but are also adaptable to future changes and can serve as a foundation for new projects. This strategic approach to software development can lead to significant cost savings and improved software longevity.

Smart Map is a software system that would support a smart tower plotting

system called OTPS that allows plotting towers on the defined map. This

Project helps in plotting the tower images on Satellite Map.

Site engineer can select particular shape and colour of the mark that will

reflect the position of a particular tower on the satellite map. It holds the

information about all the towers of operators in various site locations

which control different MSCs in company and getting tower information

by clicking on the image. In this we can create, delete and update the

tower information. This software system helps in calculating the distance

between two towers in a particular MSC. The search option will be there

for searching a particular tower by name, site identity number, latitude and

longitude. For adding information about a particular site, the Site engineer

can upload the file in a format supported by the software, which is the

indirect means of adding information regarding new sites/towers into the

database. For security reasons this project provides usernames and

passwords to control the information. Also a site engineer

can create new login ids for his/her subordinates (like trainee

etc.) who can also manage the sites in concurrently. The user

can move to any location by dragging the mouse over the

Satellite map window and user can also zoom in and zoom out

by using zoom slider.

Bugs can be in any of the 4 states - open, in progress, on hold, closed

Create a Gantt chart for this project's Implementation, testing and

deployment phase.

Draw this

Seaview hotel is a 50 room property in Mumbai. The hotel has extremely

good service at affordable prices. However as part of business expansion

plan, the management plan to develop a website. You are the project

manager in charge of the project. Identify 5 potential risk to the

development team that may arise during SDLC of the project.
Create risk

table and the mitigation activity for each risk. (add one column as mitigation activity in risk table)

Discuss the need for risk analysis in software project. Explain risk response strategies with software related example.

Identifying potential risks during the Software Development Life Cycle (SDLC) is crucial for the successful completion of a project. Here are five potential risks for the Seaview hotel website development project, along with a risk table and mitigation activities for each risk:

Risk	Description	Likelihood	Impact	Mitigation Activity
Schedule Risk	Project may exceed the planned timeline.	Medium	High	Implement agile methodologies, break tasks into smaller units, and conduct regular progress reviews.
Budget Risk	Project may go over budget due to unforeseen costs.	Low	High	Careful budget planning, regular monitoring of expenditures, and maintaining a contingency fund.
Technical Risk	Technical challenges such as integration issues or software bugs.	High	Medium	Conduct thorough testing, use continuous integration practices, and keep technical documentation up to date.

Risk	Description	Likelihood	Impact	Mitigation Activity
Operational Risk	Day-to-day operational challenges such as insufficient resources.	Medium	Medium	Effective resource management, clear communication channels, and robust process documentation.
Security Risk	Potential vulnerabilities in the website that could be exploited.	High	High	Regular security audits, adherence to security best practices, and staying updated with the latest security patches.

Need for Risk Analysis in Software Projects: Risk analysis is essential in software projects to proactively identify and address potential issues that could derail the project. It helps in:

- Prioritizing risks based on their likelihood and impact.
- Allocating resources effectively to mitigate high-priority risks.
- Developing contingency plans to handle risks if they materialize.
- Ensuring that the project stays on track, within budget, and meets quality standards.

Risk Response Strategies with Software Related Example: Risk response strategies in software development include:

- **Avoidance:** Changing the project plan to eliminate the risk or its impact. For example, if a particular technology is too new and poses a high risk, the team might choose a more established solution.
- **Mitigation:** Taking actions to reduce the likelihood or impact of the risk. For instance, implementing additional security measures to mitigate the risk of cyber attacks.
- **Transfer:** Shifting the risk to a third party, such as purchasing insurance or outsourcing a risky part of the project.
- **Acceptance:** Acknowledging the risk and not taking any action unless the risk occurs. This is typically chosen for low-priority risks.

Effective risk management is key to the success of software projects, as it ensures that risks are identified, assessed, and managed throughout the project lifecycle.

Assume you are a software project manager and that you've been asked to

compute earned value statistics for a small software project that create

widgets. The plan is to make 500 widgets with a total budget of Rs.10000

expecting to make 20 widgets per day .At the end of 5 days it is found that

70 widget were created due to training the team on the tool to be used and

the training cost is Rs.2000. Calculate CPI and SPI of the project at the

end of 5 days.

To calculate the Cost Performance Index (CPI) and Schedule Performance Index (SPI) for the project, we'll use the following formulas:

- **CPI** = Earned Value (EV) / Actual Cost (AC)
- **SPI** = Earned Value (EV) / Planned Value (PV)

First, we need to determine the Earned Value (EV), which is the value of the work actually completed. Since 70 widgets have been created and each widget is worth $\text{Rs.10000}/500 = \text{Rs.20}$, the EV is $70 \text{ widgets} * \text{Rs.20/widget} = \text{Rs.1400}$.

The Actual Cost (AC) is the actual cost incurred for the work completed. The cost so far includes the training cost plus the cost of producing 70 widgets. If we assume that the cost of production is proportional to the number of widgets produced, then the cost of producing 70 widgets is $(70/500) * \text{Rs.10000} = \text{Rs.1400}$. Adding the training cost of Rs.2000, the AC is $\text{Rs.1400} + \text{Rs.2000} = \text{Rs.3400}$.

The Planned Value (PV) is the value of the work that was planned to be completed by this time. After 5 days, the plan was to have $20 \text{ widgets/day} * 5 \text{ days} = 100 \text{ widgets}$. The PV is therefore $100 \text{ widgets} * \text{Rs.20/widget} = \text{Rs.2000}$.

Now we can calculate the CPI and SPI:

- **CPI** = Rs.1400 / Rs.3400 = 0.41
- **SPI** = Rs.1400 / Rs.2000 = 0.7

The CPI of 0.41 indicates that for every rupee spent, only Rs.0.41 worth of work was completed, which means the project is over budget. The SPI of 0.7 indicates that the project is behind schedule, as only 70% of the planned work has been completed. These metrics can help in making decisions to bring the project back on track.

Discuss software Refactoring and Reverse engineering with example.

Software Refactoring refers to the process of restructuring existing computer code without changing its external behavior. Its main purpose is to improve the nonfunctional attributes of the software. Refactoring improves the design and maintainability of the code, making it easier to understand, reducing complexity, and helping to identify and fix bugs more efficiently.

For example, consider a method in a codebase that is too long and handles multiple responsibilities. Refactoring might involve breaking this method into smaller, more focused methods that each handle a single responsibility. This makes the code cleaner, more modular, and easier to test.

Reverse Engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or a higher level of abstraction. It can be used to recover lost information, to discover the structure and operation of a program, or as part of a larger process to analyze the system for the purpose of interoperability.

For instance, if you have a legacy application for which the source code is lost or unavailable, reverse engineering can be used to analyze the compiled binaries to understand how the application works. This might involve decompiling the binaries to recover the source code or analyzing the binaries to understand their behavior and structure.

Both refactoring and reverse engineering are important practices in software development and maintenance. Refactoring helps keep the codebase clean and manageable, while reverse engineering can be essential for understanding legacy systems or integrating with third-party systems without available documentation.

Project schedule becomes a road map that defines the tasks and milestones

to be tracked and controlled.

List 5 different ways that can be used to perform tracking.

Project tracking is essential for ensuring that a project stays on schedule and within budget. Here are five different ways to perform tracking:

1. **Gantt Charts:** These provide a visual timeline for the project, showing when each task should be started and completed. They help in tracking progress against planned schedules.
2. **Critical Path Analysis:** This method identifies the sequence of crucial and interdependent steps that determine the project's duration. Tracking the critical path helps in identifying which tasks need to be prioritized to stay on schedule.
3. **Earned Value Management (EVM):** EVM is a technique that assesses project performance and progress in an objective manner by comparing the amount of work planned with what has actually been completed, and the costs incurred to accomplish it.
4. **Kanban Boards:** Kanban boards are visual tools that track the status of work items in a project. They help teams visualize workflow and identify bottlenecks in real-time.
5. **Scrum Meetings (Stand-ups):** Daily scrum meetings are short meetings used in Agile project management to ensure team members are aware of each other's progress and challenges. They provide an opportunity to adjust tasks and priorities on a daily basis.

Each of these methods offers a different approach to tracking and can be used individually or in combination to provide a comprehensive view of project progress.

Explain software change control process. List 5 reasons for software

Change

The software change control process is a systematic approach to managing modifications to a software project. It ensures that changes are made in a controlled and coordinated manner, minimizing the risk of introducing errors or disrupting the project's progress. Here's an overview of the process:

1. **Identification:** Changes are identified, often through user feedback, error reports, or new requirements.
2. **Documentation:** The proposed changes are documented, detailing the nature of the change and its potential impact.

3. **Analysis:** The change is analyzed for its implications on the project, including costs, resources, and timelines.
4. **Approval:** Changes must be approved by the relevant stakeholders or a change control board before implementation.
5. **Implementation:** Approved changes are implemented into the project.
6. **Review and Adjustment:** After implementation, the changes are reviewed to ensure they have the desired effect and adjustments are made if necessary¹.

Reasons for software change include:

1. **New Requirements:** As the business environment evolves, new requirements may emerge that necessitate changes to the software².
2. **Error Correction:** Bugs and errors discovered in the software need to be corrected to ensure it functions as intended².
3. **Performance Improvement:** Enhancements may be required to improve the performance or reliability of the system².
4. **Hardware Changes:** New computers and equipment added to the system may require software updates or changes².
5. **Regulatory Compliance:** Changes in regulations or standards may require updates to the software to ensure compliance².

These reasons highlight the dynamic nature of software development and the importance of having a robust change control process in place.

After fixing bugs in system, should the system be retested? Justify

Yes, after fixing bugs in a system, it is essential to retest the system. This process is known as **regression testing** and serves several important purposes:

- **Verify Bug Fixes:** To ensure that the bugs have been properly fixed and that the corrections work as intended.
- **Check for New Bugs:** To verify that the changes have not introduced any new bugs into the system.
- **Maintain Quality:** To maintain the overall quality and integrity of the system after changes have been made.

Retesting provides confidence that the system continues to operate correctly after modifications. It's a critical step in the software development lifecycle to ensure that the software remains stable, reliable, and performs as expected. Without retesting, there's a risk that the system might behave unpredictably or that other parts of the system might be adversely affected by the changes. Therefore, retesting is a justified and necessary practice in software maintenance and development.

A program has number of bugs equal to your 100. For testing the program, four techniques are used one after other. Each technique can detect 40% of the bugs in the system at start of testing. In addition, a bug fix can lead to new bug with a chance of 50% . Calculate the number of bugs that will remain after testing and bug fixing.

To calculate the number of bugs remaining after testing and bug fixing, we can follow these steps:

1. **Initial Bugs:** There are 100 bugs initially.
2. **Detection Rate:** Each technique can detect 40% of the bugs present at the start of its testing phase.
3. **New Bugs:** Each bug fix has a 50% chance of introducing a new bug.

Let's calculate the number of bugs remaining after each testing technique:

- **After 1st Technique:** 40% of 100 bugs are detected and fixed, leaving 60 bugs. However, 50% of the 40 fixed bugs introduce new bugs, so 20 new bugs are added, resulting in $60 + 20 = 80$ bugs.
- **After 2nd Technique:** 40% of 80 bugs are detected and fixed, leaving 48 bugs. Again, 50% of the 32 fixed bugs introduce new bugs, so 16 new bugs are added, resulting in $48 + 16 = 64$ bugs.
- **After 3rd Technique:** 40% of 64 bugs are detected and fixed, leaving 38.4 bugs. 50% of the 25.6 fixed bugs introduce new bugs, so 12.8 new bugs are added, resulting in $38.4 + 12.8 = 51.2$ bugs.
- **After 4th Technique:** 40% of 51.2 bugs are detected and fixed, leaving 30.72 bugs. 50% of the 20.48 fixed bugs introduce new bugs, so 10.24 new bugs are added, resulting in $30.72 + 10.24 = 40.96$ bugs.

After four rounds of testing and bug fixing, approximately **41 bugs** will remain in the system. It's important to note that in practice, the number of bugs would be a whole number, so this would be rounded to the nearest whole number. Additionally, the probability of introducing new bugs might change as the system evolves, and the effectiveness of testing techniques might also vary. This calculation assumes a constant rate for simplicity.

1. Before testing starts, a program is known to contain 1000 bugs. During testing of the program, four testing techniques are used in succession. Each testing technique is effective in detecting 60% of the bugs present at the time of starting that testing. Further, there is a 50% chance that a bug fix gives rise to a new bug. Approximately how many bugs would remain after application of the four bug detection techniques and correction of the detected bugs?
- a. 150
 - b. 200
 - c. 250
 - d. 300

Explanation: Because of the first testing technique, number of bugs that remain is $(1 - 0.6) * 1000$ (bugs undetected) + $0.5 * 0.6 * 1000$ (bug triggered because of the bug fixing technique) i.e. $1000 * (0.4 + 0.3) = 0.7 * 1000$.

So, after four similar testing techniques, the number of bugs that remain is approximately $(0.7 * 0.7 * 0.7 * 0.7) * 1000 \sim 250$ bugs.

Jan 2020

Explain various steps involved in SRS with suitable example

The Software Requirements Specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also details the functionality the product needs to fulfill all stakeholder needs. Here are the steps involved in creating an SRS, with an example:

1. **Introduction:** This section outlines the purpose, scope, and overview of the document. It sets the stage for the rest of the SRS by explaining what the document will cover and why it's necessary¹.

Example: For a project management software, the introduction would explain the need for such a tool, its intended users, and the problems it aims to solve.

2. **General Description:** This part provides a general overview of the software, including user characteristics, constraints, assumptions, and dependencies¹.

Example: The general description for a healthcare app might include the types of users (patients, doctors), regulatory constraints, and assumptions about user technical skills.

3. **Functional Requirements:** This crucial section details the specific functionalities the software must have. It includes use cases, data processing, and precise descriptions of each function¹.

Example: A functional requirement for an e-commerce platform could be the ability for users to filter products by category, price, and brand.

4. **Interface Requirements:** Here, the interfaces between the software and other systems or users are described. It covers user interfaces, hardware interfaces, software interfaces, and communication interfaces¹.

Example: For a mobile banking app, interface requirements would describe the app's layout, integration with payment gateways, and communication protocols with the bank's servers.

5. **Performance Requirements:** This section specifies the performance benchmarks the software must meet, such as response times, throughput rates, and data capacity¹.

Example: A video streaming service's performance requirements might include the ability to support a certain number of concurrent streams with specific loading times.

6. **Design Constraints:** Any constraints on the system design, such as standards compliance, hardware limitations, or language requirements, are listed here¹.

Example: A software developed for government use might need to comply with specific security standards and support legacy hardware.

7. **Non-Functional Requirements:** These requirements cover aspects not directly related to functionality, such as security, reliability, maintainability, and portability¹.

Example: Non-functional requirements for a cloud storage service could include data encryption standards, uptime guarantees, and support for multiple operating systems.

8. **Preliminary Schedule and Budget:** This part may include a high-level schedule for the project's completion and a budget estimate¹.

Example: For a custom CRM system, the preliminary schedule might outline major milestones, and the budget would estimate the cost for development, testing, and deployment.

9. **Appendices:** Any additional information that supports the SRS can be included in the appendices. This might consist of glossaries, references, or supporting documents¹.

Example: Appendices for a gaming app might include a glossary of gaming terms and references to similar apps for context.

Creating an SRS is a collaborative effort that requires input from stakeholders, users, and the development team. It serves as a blueprint for the project and ensures that everyone has a clear understanding of the software's objectives and requirements. The SRS helps to minimize misunderstandings and scope creep by providing a detailed and agreed-upon set of requirements before development begins.

Explain in detail incremental model with advantages and disadvantages. Give one example of using of Incremental model.

The Incremental Model in software engineering is a development approach where the product is designed, implemented, and tested incrementally until the product is finished. It allows for partial implementation of the product, which can then progressively gain more functionality until it is complete. Here's a detailed explanation along with its advantages, disadvantages, and an example:

Advantages:

- **Flexibility:** Allows for changes to be made after the initial stages of development¹.
- **Risk Management:** High-risk features can be developed earlier, which helps in risk management².
- **Customer Feedback:** Customers can provide feedback on each increment, allowing for adjustments before the next increment is developed¹.
- **Early Partial Deployment:** Parts of the system can be deployed and used before the entire system is complete².
- **Testing and Debugging:** Easier to test and debug during a smaller iteration¹.

Disadvantages:

- **Planning:** Requires good planning and design¹.
- **Total Cost:** The total cost may be higher compared to other models due to the iterative nature of the process¹.
- **Well-Defined Interfaces:** Requires well-defined interfaces between increments².
- **System Architecture:** The system architecture may need to be defined fully from the beginning, which can limit flexibility in some cases².

- **Integration:** Requires careful attention to the integration of increments².

Example: An example of using the Incremental Model is in the development of a web application. The first increment might include the basic user interface and login functionality. Once this is deployed and feedback is received, the next increment might add user profile management. Subsequent increments could introduce features like search functionality, messaging, and notifications. This approach allows the development team to focus on one set of functionalities at a time, ensuring quality and allowing for user feedback to guide future increments.

The Incremental Model is particularly useful when the requirements are well-understood and there's a need to get a product to market quickly with the most essential features, while additional features can be added over time¹². It's a balance between the flexibility of Agile and the rigidity of the Waterfall model.

Explain in detail Prototyping model with advantages and disadvantages. Give one example of using of Prototyping model.

The **Prototyping Model** in software development is an iterative, trial-and-error process that involves creating prototypes of a software application before building the final product. This model is particularly useful when the requirements are not well-understood or are expected to change. Here's how it works and its advantages and disadvantages:

Steps in the Prototyping Model:

1. **Requirement Gathering and Analysis:** Collecting the initial requirements from the user to understand their expectations¹.
2. **Quick Design:** Creating a preliminary design of the system¹.
3. **Build a Prototype:** Developing a working model of the software based on the quick design¹.
4. **Initial User Evaluation:** The prototype is evaluated by the user, and feedback is collected¹.
5. **Refining Prototype:** Based on the feedback, the prototype is refined and improved¹.
6. **Implement Product and Maintain:** Once the prototype meets the user's needs, the final system is developed and maintained¹.

Advantages:

- **User Involvement:** Users are involved early in the development, ensuring the final product meets their needs¹.
- **Requirement Clarification:** Helps in clarifying requirements which are not well understood at the beginning¹.

- **Reduced Risk of Failure:** By testing prototypes, the risk of failure in the final product is reduced¹.
- **Quick Feedback:** Allows for quick user feedback and iterative refinement¹.

Disadvantages:

- **Scope Creep:** Continuous changes can lead to scope creep, where the project keeps expanding beyond its original objectives¹.
- **Cost and Time:** Can be more costly and time-consuming due to the iterative nature of the process¹.
- **Incomplete Solution:** Early prototypes might give an incomplete picture of the system's capabilities¹.
- **Excessive Development Time:** Too much time spent on developing prototypes can delay the final product¹.

Example of Prototyping Model: A classic example of using the Prototyping Model is in the development of a new customer relationship management (CRM) system for a sales department. The initial prototype might include basic features like contact management and lead tracking. As the sales team uses the prototype, they provide feedback, leading to new iterations that might add advanced features such as sales forecasting, integration with email marketing, and custom reporting. This process continues until a satisfactory system is developed that meets all the sales team's needs.

The Prototyping Model is particularly beneficial in scenarios where user requirements are not fully understood or are likely to evolve, as it allows for flexibility and user-driven design throughout the development process¹.

Illustrate with example different types of coupling

Coupling in software engineering refers to the degree of interdependence between software modules. A lower degree of coupling is generally preferred as it indicates that modules are more independent, making the system easier to maintain and modify. Here are different types of coupling with examples:

1. **Data Coupling:** Occurs when modules share data through parameters. For example, if a function `calculateTotal` takes individual prices as parameters and returns the total price, it exhibits data coupling¹.
2. **Stamp Coupling:** Happens when modules share a composite data structure, such as a class or a struct, but only use a part of it. For instance, if a function `processOrder` takes an `Order` object but only uses the `orderDate` field, it's an example of stamp coupling¹.

3. **Control Coupling:** Involves one module controlling the behavior of another by passing it information on what to do. [For example, a function `displayMessage` that takes a boolean flag to determine whether to display a message in uppercase or lowercase is control coupled¹.](#)
4. **External Coupling:** Arises when modules are dependent on external systems or conventions. [For example, if a module interacts with a specific database or file format, it has external coupling¹.](#)
5. **Common Coupling:** Occurs when multiple modules have access to the same global data. [For example, if several functions read and write to a global configuration object, they are commonly coupled¹.](#)
6. **Content Coupling:** The strongest and least desirable type of coupling, content coupling happens when one module directly modifies or relies on the internal workings of another module. [For example, if a function A directly changes the variables within function B, it is content coupled¹.](#)

Each type of coupling has its implications on the design and maintainability of a software system, with lower coupling generally leading to a more robust and flexible design.

Explain RMMM plan in detail

The Risk Mitigation, Monitoring, and Management (RMMM) plan is a comprehensive strategy used in software engineering to manage risks throughout the lifecycle of a project. It is typically included within the software project plan and can be divided into three main activities: Risk Mitigation, Risk Monitoring, and Risk Management. Here's a detailed explanation of each:

Risk Mitigation:

- **Finding out the risk:** Identifying potential risks that could affect the project.
- **Removing causes:** Taking steps to eliminate the factors that contribute to these risks.
- **Controlling documents:** Regularly reviewing and controlling documentation to prevent risks associated with outdated or incorrect information.
- **Conducting reviews:** Holding timely reviews to expedite work and address risks early on.

Risk Monitoring:

- **Tracking predicted risks:** Checking if the anticipated risks occur.
- **Ensuring risk aversion steps:** Making sure that the defined steps for avoiding risks are properly applied.
- **Collecting data:** Gathering data for future risk analysis and understanding the impact of risks on the project.

- **Problem allocation:** Determining which problems are caused by which risks throughout the project.

Risk Management:

- **Planning for risks:** Assuming that mitigation activities might fail and preparing for the possibility that risks become a reality.
- **Managing active risks:** Dealing with risks that have materialized and are causing issues in the project.
- **Risk register:** Maintaining a risk register that describes and focuses on the predicted threats to the project.

An example of RMMM in action could be managing the risk of high staff turnover. In this case, risk mitigation might involve strategies to reduce turnover, such as improving working conditions or offering competitive salaries. Risk monitoring would include keeping an eye on staff satisfaction levels and turnover rates, while risk management would involve having contingency plans in place, such as cross-training team members to ensure continuity of work¹.

The RMMM plan is crucial for proactive risk management in software projects, helping teams to anticipate, address, and navigate potential challenges effectively¹.
It is a dynamic document that evolves as the project progresses and as new risks are identified and managed.

Explain different types of user interfaces

- Q.2 a) Types of user interfaces with explanation –6M
- 1) Command language-based interfaces
 - 2) Menu- based interfaces
 - 3) Direct manipulation interfaces

<https://alan.app/blog/types-of-user-interface/>

<https://www.techopedia.com/definition/3337/command-line-interface>
cli#:~:text=Techopedia%20Explains%20Command%20Line%20Interface&text=CLI%20is%20a%20text%2Dbased,it%20is%20not%20user%20friendly.

User interfaces (UI) are the spaces where interactions between humans and computers occur. The goal is to make these interactions efficient, allowing

machines to aid in decision-making and users to operate them effectively. Here are different types of user interfaces:

- **Graphical User Interface (GUI):** Uses visual representations like windows, icons, and menus. It's suitable for non-technical users and provides immediate visual feedback, but it requires more power and memory resources¹.
- **Touchscreen Graphical User Interface:** Similar to GUI but designed for touch input. It supports gestures like swiping and tapping, commonly used in mobile devices¹.
- **Menu-Driven Interface:** Users navigate through a system using a list of options or menus. This type is often found in ATMs and is simple to use².
- **Command Line Interface (CLI):** Users interact with the software using text-based commands. It's powerful for experienced users but has a steeper learning curve².
- **Form-Based Interface:** Users enter data into fields or select options in a form. This type is commonly used for settings menus and web forms².
- **Voice User Interface (VUI):** Allows users to interact with a system through voice commands. Examples include virtual assistants and GPS devices².
- **Conversational User Interface (CUI):** Mimics human conversation, either through text or voice. Chatbots are a common example¹.
- **Natural User Interface (NUI):** Focuses on human actions that come naturally, like touch, gestures, and speech³.
- **Virtual Reality (VR) and Augmented Reality (AR) Interfaces:** Provide immersive experiences by integrating digital information with the user's environment in real-time³.

Each type of UI has its own set of advantages and challenges, and the choice of UI depends on the specific needs and context of the application. Designing a good UI requires understanding the target users and the tasks they need to perform to ensure the interface is intuitive and efficient¹².

What do you understand by system testing?

Alpha,beta and acceptance testing



Explore

System testing is a level of software testing where a complete and integrated software system is tested to evaluate the system's compliance with its specified requirements. [It is typically performed after integration testing, which tests the interactions between different software modules, and before acceptance testing, which is the final testing phase before the software is released to end-users¹.](#)

Alpha Testing is a type of acceptance testing performed by testers who are usually internal employees of the organization developing the software. It is conducted in a lab environment and focuses on identifying bugs and issues before the software is released to external users. [Alpha testing is often the last testing stage before beta testing and involves both white box and black box testing techniques¹².](#)

Beta Testing is also a type of acceptance testing but is performed by a group of end-users. It takes place in a real-world environment and aims to uncover bugs that may not have been identified during alpha testing. Beta testing helps in validating the software's performance and functionality from the user's perspective. [It is one of the final steps before the software is fully released to the public³.](#)

Acceptance Testing, as mentioned earlier, is the final phase of software testing. It is performed to ensure that the software meets the business requirements and is ready for delivery. Acceptance testing can be further divided into different types, such as User Acceptance Testing (UAT), Business Acceptance Testing (BAT), Contract Acceptance Testing (CAT), and Regulation Acceptance Testing (RAT). [Each type focuses on different aspects, such as meeting user needs, achieving business goals, fulfilling contract obligations, and complying with regulations, respectively⁴.](#)

These testing phases are crucial for ensuring that the software is of high quality, meets user expectations, and functions correctly in the intended environment. They are integral parts of the software development lifecycle and contribute significantly to the success of the software product.

What are the different types of testing that are usually performed on large scale software projects

In large-scale software projects, several types of testing are typically performed to ensure the software's quality and functionality. Here are some of the common types of testing used:

- **Unit Testing**: Tests individual units or components of the software to ensure they function correctly in isolation¹.
- **Integration Testing**: Ensures that different modules or services used by the application work well together¹.
- **System Testing**: Validates the complete and integrated software to check whether it meets the specified requirements¹.
- **Functional Testing**: Checks the software against the functional requirements to ensure all features work as expected¹.
- **Acceptance Testing**: Also known as User Acceptance Testing (UAT), it determines if the software is ready for release by validating it against user needs and requirements¹.
- **Smoke Testing**: A preliminary test to check the basic functionality of the software before it goes into detailed testing¹.
- **Regression Testing**: Performed after changes to the software to ensure that the new code has not affected existing functionality¹.
- **Performance Testing**: Tests the software's performance under various conditions to ensure it meets performance criteria¹.
- **Security Testing**: Identifies vulnerabilities in the software to ensure that data and resources are protected from potential threats¹.
- **Usability Testing**: Evaluates the software's user interface and user experience to ensure it is user-friendly and intuitive¹.
- **Compatibility Testing**: Checks the software's compatibility with different browsers, databases, hardware, operating systems, mobile devices, and networks¹.
- **Load Testing**: Measures the system's performance under a specific expected load¹.

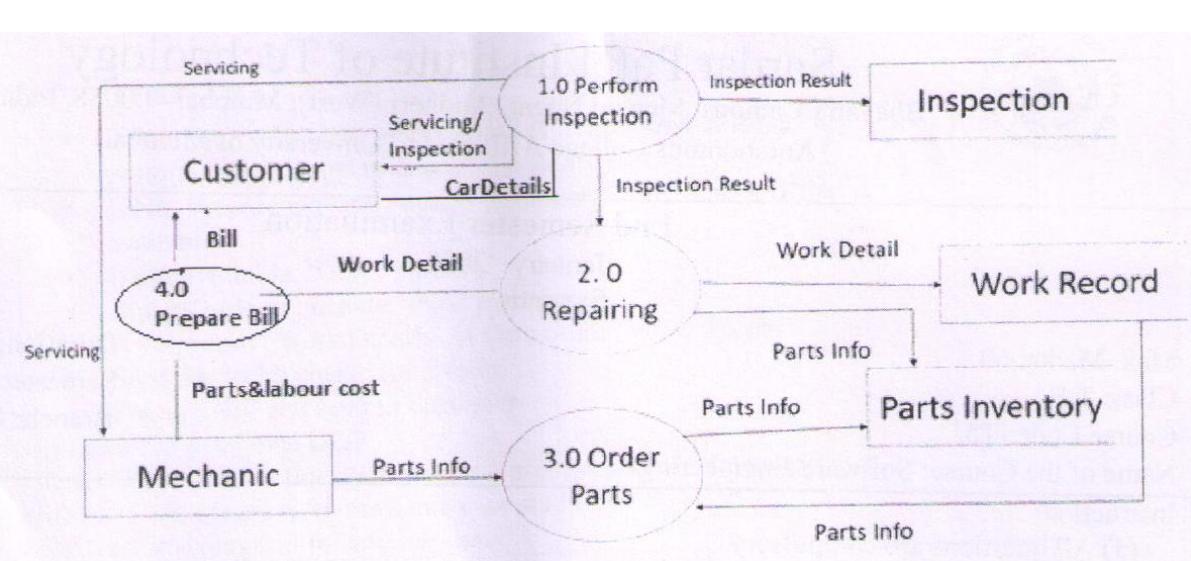
These types of testing help identify issues early in the development process, ensuring that the software is reliable, secure, and user-friendly before it is deployed to users¹. Each type of testing serves a specific purpose and helps in building a robust and high-quality software product.

Q.3 a) Event List for the case study: Garage system is given below

- Customer comes for vehicle check-up.
- Inspection is performed.
- Is there some problem or just needs some servicing.
- If Nothing, prepare Inspection Result and send to Data Store 'Inspection' and also prepare Bill and send to customer
- If Servicing, Send to Mechanic for servicing. Prepare Inspection Result and send to Data Store 'Inspection' and also Prepare Bill and send to customer

- If Part faulty, Prepare Inspection Result and send to Repairing Ask customer if he wants to change the faulty part
- If No, Prepare Bill and send to customer. If Yes, check if Part is available.
- If not available, 'Order parts'.
- If parts ordered, update 'Parts Inventory'
- If yes, input 'Work Details' in Data Store 'Work Record'.
- Send car to Mechanic for Repair.
- Prepare Bill and send to customer

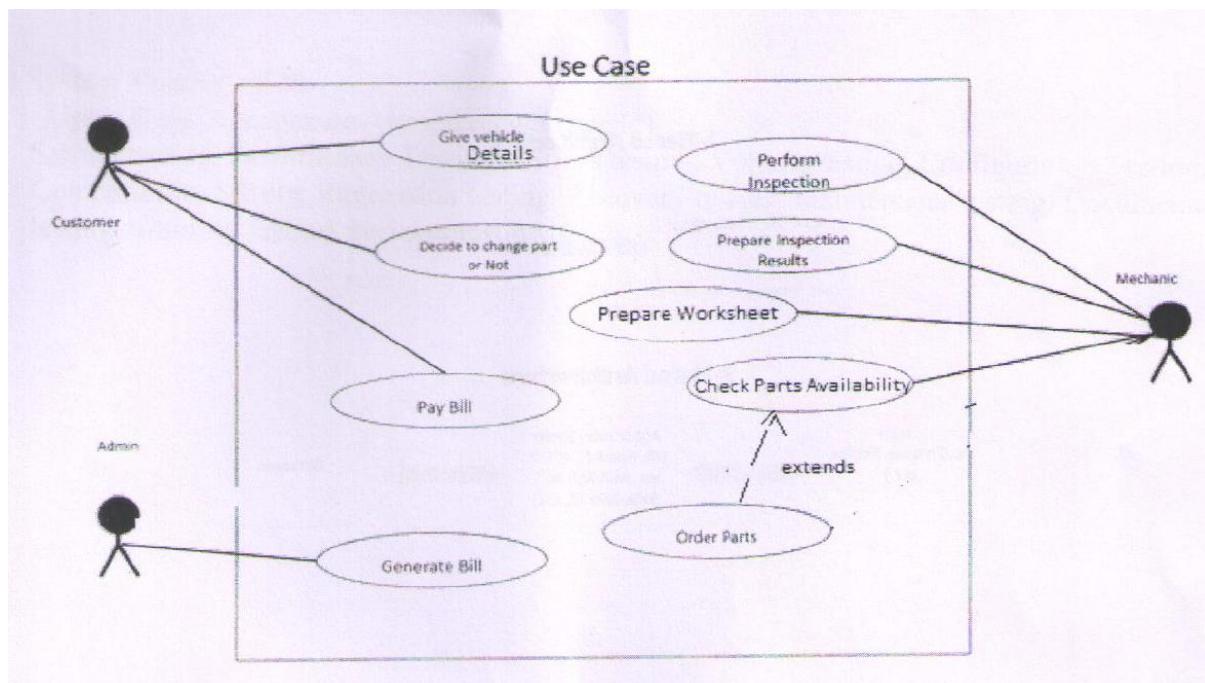
- (i) Draw the first level DFD
- (ii) Write the data dictionary entry for 1 process and one data store
- (iii) Use Case diagram for the same case study
- (iv) Sequence diagram for any use case



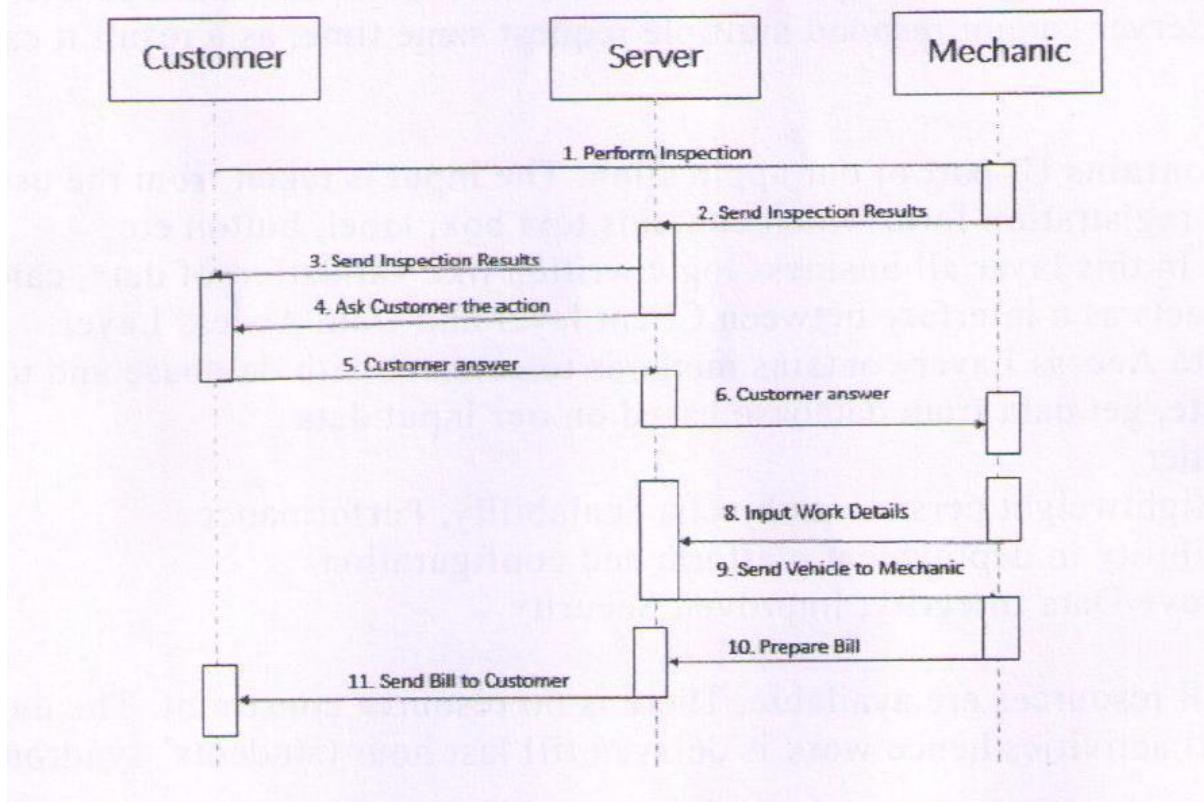
(ii) DD

Data Store	
Name	Work Record
Description	Data Store where work details are saved
Elements	Record No., Work Detail
Access Rights	Admin
Volume	1GB
Master/Transaction	Transaction

Process	
Name	Check Part Availability
Description	Check if Parts are available in Parts Inventory
Inbound Data Flow	Faulty part name
Outbound Data Flow	Customer Decision
Process Logic	If Customer does not want to 'change part' then 'Prepare Inspection Bill' If Customer wants to 'change part' then give 'change' to "Input work Detail"



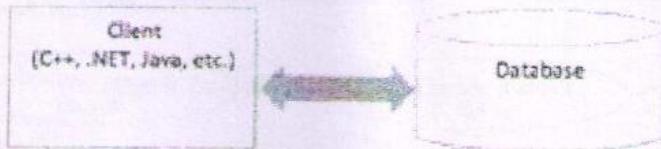
Sequence Diagram



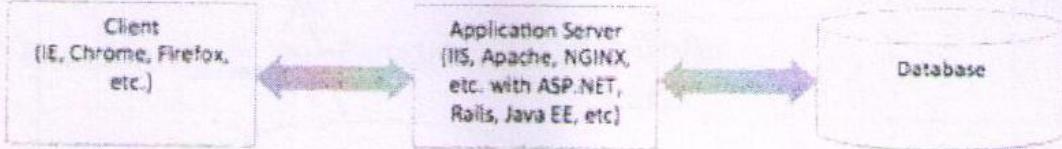
(iv) Sequence diagram for any use case

Draw the architecture diagram for Railway Reservation as 2 tier (client-server) and 3-tier architecture and discuss

2-Tiered Architecture



3-Tiered Architecture



Disadvantages (Cons) of Critical Path Method (CPM) are overcome by Critical Chain Project Management (CCPM) method-Justify

4b) CPM assumes all resources are available. There is no resource constraint. The duration taken includes buffer in all activities hence work is delayed till last hour (students' syndrome). So project delay is possible

In CCPM the resources which are used in the critical chain activities are usually considered as critical resources. Activities that are not included in the critical chain while at the same time converging to critical chain are considered feeders. The main focus of CCPM is to eliminate the uncertain delays, task overestimation duration delays, and wasted internal buffers delays. In CCPM, project duration does not change even if all the activity safety margins were eliminated, because of the project buffer. Project buffer protects the project completion on the critical chain path, while feeding buffers protects the critical chain from path merging. Managing the buffer further improve the decision making of project control.

MSE OCT 22

Q.1 B Draw Data Flow Diagram) for Railway Reservation System. (Level 0, Level 1, and Level 2 for any one process of level 1

For the given scenario of a banking customer applying for loan, draw sequence diagram. Show synchronous, asynchronous and return message.

A customer gives the application for loan to the bank .

The bank teller sends the application to be processed by the bank manager and waits for the manager to approve/reject loan.

The bank manager sends data to the credit agency for verification.

The bank manager receives a response and approves or rejects loan.

The bank teller sends a message to the customer about whether the loan was approved or rejected.

Use line with a solid arrowhead to represent synchronous message and line with open/simple arrowhead to represent asynchronous message.

Q.2 B	What are 2 aspects of a software system that are explicitly omitted from the UML class diagram but captured in sequence diagram or state chart diagram? (only 2 sentences) Model the following with UML class diagram. a) During one soccer season, multiple players participate in multiple games. Each player scores in each game a certain number of goals. b) Every restaurant has at least one kitchen, one kitchen is part of exactly one restaurant.
--------------	---

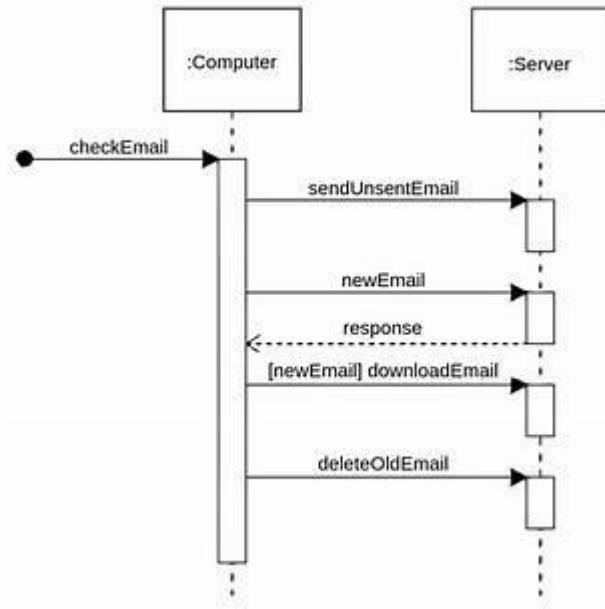
What is the difference between requirement analysis and specification?

Requirement analysis and specification are two critical phases in the software development process, each serving a distinct purpose:

- **Requirement Analysis:** This phase focuses on understanding and capturing the needs and expectations of stakeholders. It involves activities such as requirements gathering, where information is collected from stakeholders, and requirements analysis, where the collected data is examined to conceptualize what exactly needs to be done. The goal is to understand the precise user requirements and analyze them to identify any inconsistencies, anomalies, and incompleteness¹.
- **Specification:** After the requirements are analyzed, the specification phase involves transforming those needs into a detailed document that guides the software development process. This document, usually called the Software Requirements Specification (SRS), outlines what the software should do and how it should perform, without detailing how the software will be implemented².

In summary, requirement analysis is about understanding ‘what’ the stakeholders need, while specification is about defining ‘how’ to achieve those needs in a document that will guide the development team³. The specification serves as a bridge between the initial understanding of the user’s needs and the technical design that will fulfill those needs⁴.

Define asynchronous and self message in sequence diagram with one example each.



An **asynchronous message** in a sequence diagram represents a communication between objects where the sender does not wait for a response before continuing its activity. [It's depicted with a lined arrowhead and is used to model non-blocking calls¹.](#)

For example, in a chat application, when a user sends a message, the system might asynchronously notify the recipient without waiting for any acknowledgment that the message has been read.

A **self message**, also known as a self-call, occurs when an object calls a method on itself within the same lifeline. [It's represented by an arrow that points back to the same lifeline, often with a nested activation bar to show the method invocation².](#)

For example, in a graphics program, an object representing a shape might have a method to calculate its area. The `calculateArea` method might call another method on the same object, such as `updateDisplay`, to refresh the shape's appearance on the screen after the area is calculated.

March 2020

Given process models

A)Waterfall B)Incremental C)RAD Model D)Spiral
and properties of process models

E) Angular component represent progress

F) Porting an existing product to a new platform

G) Need to get basic functionality early in market

H) Increases reusability

I) Open source projects

J) Uses “divide and conquer” breakdown of task

Relate 1 or more properties to the given process models

List 3 reasons why a system should be analyzed before it is implemented?

Angular component represent progress - Spiral

Porting an existing product to a new platform – Waterfall

Need to get basic functionality early in market – Incremental

Increases reusability – Rad Model

Open source projects – Rad Model

Uses “divide and conquer” breakdown of task – Incremental

It is necessary to establish the system boundaries which would define the scope of the system.

To seek the acceptance of the people to a new development.

The analysis of the system helps to establish the feasibility from different angles. The system should satisfy the technical, economic and operational feasibility.

Identify if given user story is good or bad. Justify.

a)As a user I want the system to use log4j to log all the messages so as to detect errors.

b)As a user I want to export data to XML so as to use data in XML format.

ANSWER

As a user I want the system to use log4j to log all the messages.

Not a good user story as it is part of the developer task, what mechanism to use

As a user I want to export data to XML so as to use data in XML format

Good user story

Discuss SCRUM process and the artifacts, roles, ceremonies in SCRUM

SCRUM is iterative & incremental agile software development method. These iterations are time boxed with various iterations & each iteration is called Sprint. The Sprint is basically 2-4 week long & each sprint requires sprint planning estimation.

3 Roles

Product Owner, SCRUM Master, Developer

3 Artifacts

Product Backlog

Sprint Backlog

Burndown Chart

3 Ceremonies

Sprint Plan

Daily Scrum – Stand up meetings

Sprint Review – Team meeting with product owner Discusses what to do to improve process

Imagine two flight ticket reservation systems, each of them written in a different programming language. One consists of ten thousand lines of

code, and another one has two thousand lines of code. Does it mean the first system is more complex than the second? Suggest best way to compare the 2 systems. Justify.

No, more number of lines of code does not indicate more complexity.
 The FPA method can be used to give realistic comparison of 2 applications.
 Because FPA estimation is independent of language.

How is Earned Value Analysis used to track a project? Discuss with an example.

ANSWER

Planned Value (PV): It is the approved budget for the work to be done for an activity or work package (or control account). T

Earned Value (EV): Formally, it was called the Budgeted Cost of Work Performed or BCWP. EV is the value of work performed in terms of authorized budget for that work element(s).

Actual Cost (AC): Formally, it was called the Actual Cost of Work Performed (ACWP). AC is the value of the actual cost incurred or realized for the performed work during a particular time period.

Cost Variance (CV) = EV – AC

Schedule Variance (SV) = EV – PV

Negative value of CV & SV indicate it is over budget, over schedule

Given the following values compute the F.P when all complexity adjustment factors, and weighting factors are average.

User input= 50, user output= 40, user enquiries= 35, user files = 6, external interface= 4

Unadjusted function point scale:

0: No influence, 1: Incidental, 2: Moderate, 3: Average, 4: Significant, 5: Essential

Functional units	Weighting factor		
	Low	Average	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

What is FTR? Explain responsibility of participants in a FTR meeting

Formal Technical Review (FTR) is a software quality control activity performed by software engineers. The objectives of FTR are to uncover errors in logic, function, and implementation for any representation of the software, verify that the software meets specified requirements, ensure that software is represented according to predefined standards, review the uniformity in software development, and make the project more manageable¹.

The responsibilities of participants in an FTR meeting typically include:

- **Review Leader:** Plans the review, selects the team, and leads the meeting. Ensures that the review process is followed and that all issues are documented¹.
- **Author:** The creator of the work product being reviewed. Answers questions and clarifies any doubts about the product¹.
- **Reviewers:** They are responsible for examining the work product before the meeting and identifying potential issues. During the meeting, they discuss these issues and provide constructive feedback¹.
- **Scribe:** Records all issues raised during the review and ensures that a review issue list is prepared at the end of the meeting¹.
- **Manager:** Although not always directly involved in the review, the manager may attend to understand the issues and follow up on the resolution of critical problems¹.

Each participant has a specific role to play to ensure the effectiveness of the FTR process. The review is considered successful only if it is properly planned, controlled, and attended¹.

Explain different costs incurred to maintain quality of product

Maintaining the quality of a product involves various types of costs, which can be categorized into four main components:

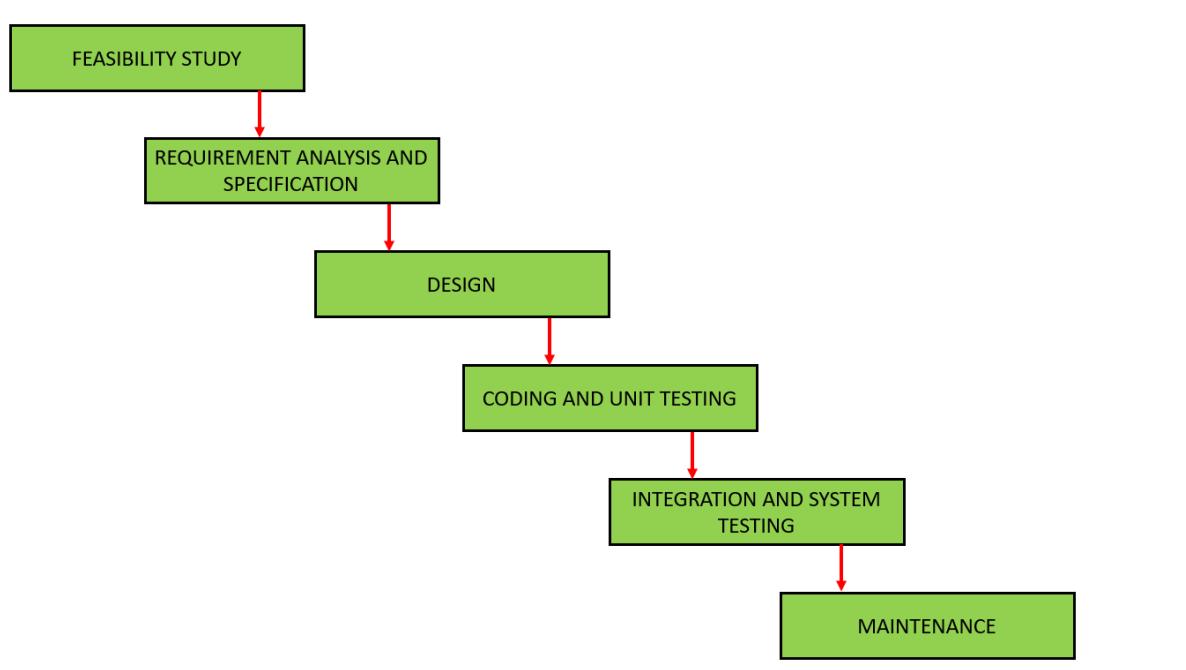
1. **Prevention Costs:** These are costs incurred to prevent defects from occurring in the first place. They include activities such as quality planning, training, and process control measures to ensure that the product is designed and developed correctly from the beginning¹.

2. **Appraisal Costs:** These costs are associated with measuring and monitoring activities related to quality. They include inspection, testing, and audits to ensure that the product conforms to quality standards¹.
3. **Internal Failure Costs:** These are costs related to defects found before the product reaches the customer. They include waste, scrap, rework, and failure analysis. Internal failure costs arise when the results of work fail to reach design quality standards and are detected before being shipped to the customer¹.
4. **External Failure Costs:** These costs occur when defects are found after the product has been delivered to the customer. They include warranty claims, returns, and repairs. External failure costs are often significant because they not only include the cost of fixing the defect but also handling customer complaints and lost goodwill¹.

By understanding and managing these costs, organizations can improve their quality processes, which can lead to a reduction in the total cost of quality over time. It's important for companies to invest in prevention and appraisal costs to reduce the likelihood and impact of internal and external failures².

Re-exam Jan 2020

Identify and elaborate phases of waterfall model



1. Feasibility Study

The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.

2. Requirements Analysis and Specification

The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

- **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
- **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. Design

The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A [Software Design Document](#) is used to document all of this effort (SDD)

4. Coding and Unit Testing

In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. Integration and System testing

Integration of different modules is undertaken soon after they have been coded and unit tested. Integration of various modules is carried out

incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this. System testing consists of three different kinds of testing activities as described below.

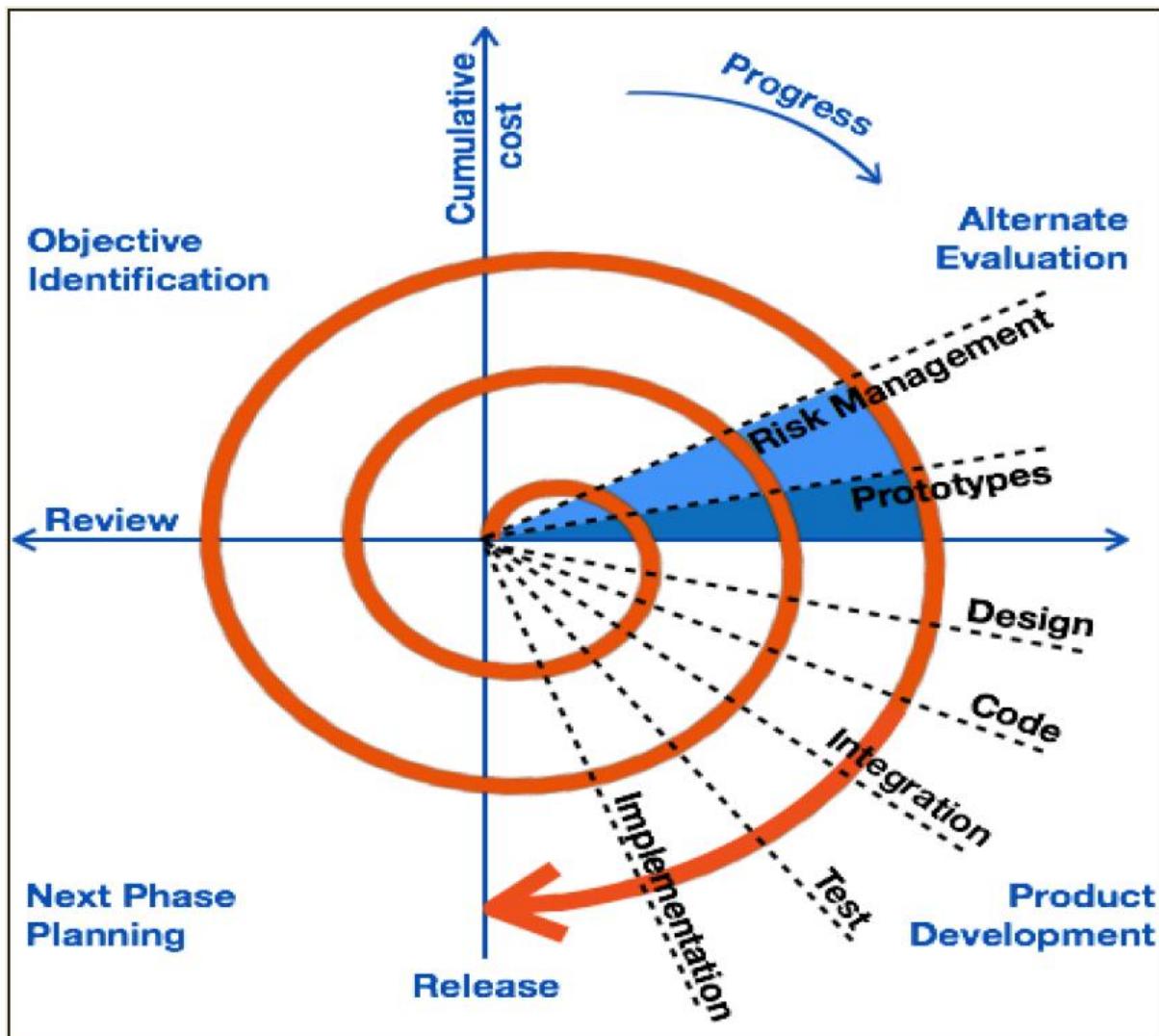
- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

6. Maintenance

Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are basically three types of maintenance.

- **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
- **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Identify and elaborate phases of Spiral model



First quadrant (Objective Setting)

- During the first quadrant, it is needed to identify the objectives of the phase.
- Examine the risks associated with these objectives.

Second Quadrant (Risk Assessment and Reduction)

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

Third Quadrant (Development and Validation)

- Develop and validate the next level of the product after resolving the identified risks.

Fourth Quadrant (Review and Planning)

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

Why Q & A is recommended in SDLC process??

Quality Assurance (QA) is recommended in the Software Development Life Cycle (SDLC) process for several important reasons:

1. **Preventive Approach:** QA is proactive, aiming to identify and avoid potential issues before they occur, rather than being reactive¹.
2. **Enhanced Product Quality:** QA ensures that the software meets the planned functionality, is bug-free, and works flawlessly under different circumstances².
3. **Cost Efficiency:** Engaging QA early in the SDLC can save significant amounts of money by avoiding the high costs associated with fixing defects later in the development process¹.
4. **User Experience:** QA testers are advocates for user experience, ensuring that the software is user-friendly and meets customer expectations¹.
5. **Risk Mitigation:** Proper QA processes help to find and eliminate possible security and performance issues before the product is released, reducing technical and commercial risks³.
6. **Continuous Improvement:** QA is an integral part of all phases of the SDLC, contributing to continuous improvement and higher quality outcomes throughout the project⁴.
7. **Better End Product:** By fully integrating QA into the SDLC, the quality of the final product is improved, leading to more satisfied customers and potentially higher profits⁵.

In summary, QA is essential for building a reliable, secure, and stable product that matches expectations and meets end users' needs. **It's not just about testing; it's about ensuring overall quality throughout the entire development process³.**

Which methods will you use to collect requirements from customers

To collect requirements from customers, there are several methods that can be effectively used. Here are some of the methods based on the search results:

1. **Interviewing**: This involves meeting with stakeholders through meetings, phone calls, or emails to gather their requirements¹.
2. **Surveys/Questionnaires**: Distributing surveys or questionnaires to customers to collect their feedback and requirements².
3. **Focus Groups**: Engaging a group of people in discussions to gain insights into their needs and expectations².
4. **Workshops**: Conducting interactive sessions where stakeholders can collaborate to define requirements³.
5. **Observation**: Watching how users interact with current systems or products to identify potential improvements⁴.
6. **Document Analysis**: Reviewing existing documentation and reports to extract requirements⁵.
7. **Prototyping**: Creating a preliminary version of a product to gather feedback and refine requirements³.
8. **Brainstorming**: Generating a wide range of ideas and solutions through group discussion³.

Each method has its own advantages and can be chosen based on factors such as the project's complexity, stakeholder availability, and the nature of the requirements to be collected. It's often beneficial to use a combination of these methods to ensure a comprehensive collection of customer requirements.

Differentiate between COCOMO and COCOMO II in detail.

Certainly! COCOMO (Constructive Cost Model) and COCOMO II are both software cost estimation models developed by Barry W. Boehm, but they have some key differences:

COCOMO:

- Also known as Basic COCOMO, it was the first model introduced.
- It is suitable for traditional waterfall models of software development.
- The model estimates effort, cost, and schedule based on lines of code.
- It assumes reasonably stable software requirements.
- The effort equation's exponent is determined by three development modes: Organic, Semi-Detached, and Embedded.
- It uses a linear reuse formula for calculations.

- COCOMO has three submodels and 15 cost drivers.

COCOMO II:

- It is the revised version of the original COCOMO model.
- It is designed for non-sequential, rapid development, and reuse models of software.
- The model estimates development time and effort as the total of the estimates of all individual subsystems.
- It is based on a non-linear reuse formula.
- COCOMO II accounts for the effort needed to understand and estimate reused components.
- The effort equation's exponent is determined by five scale factors.
- It has four submodels and 17 cost drivers.
- The size of the software can be stated in terms of object points, function points, and lines of code.

In summary, COCOMO II is more adaptable to modern software development practices, including rapid development and reuse, and provides a more detailed approach to cost estimation with additional factors and submodels compared to the original COCOMO model¹².

Discuss three levels of model of and development modes of COCOMO model in detail

The COCOMO (Constructive Cost Model) developed by Barry Boehm is a software cost estimation model that has three levels of models and three development modes. Here's a detailed look at each:

Three Levels of COCOMO Model:

1. **Basic COCOMO Model:** This is the simplest form of the COCOMO model and is used for quick, rough order of magnitude estimates of software costs. It calculates the effort and cost of a project based on the size of the software (measured in Kilo Lines of Code - KLOC) and a set of constants that vary based on the project type¹.
2. **Intermediate COCOMO Model:** This level includes the features of the basic model and introduces cost drivers that account for differences in hardware constraints, personnel quality, use of modern tools, and other project attributes. It provides more accurate estimates by considering these additional factors¹.
3. **Detailed COCOMO Model:** The most detailed level, it breaks down the software into smaller components and estimates the cost of each one separately. This level considers all characteristics of the intermediate model

and also includes an assessment of the impact of each cost driver over the different stages of the project¹.

Development Modes of COCOMO Model:

1. **Organic Mode:** This mode applies to projects with “small” teams working in a “stable” environment on projects with “relatively small” size. It is used for projects where the team has experience with similar projects, and the requirements are well understood²¹.
2. **Semi-Detached Mode:** This mode is for projects that fall between organic and embedded modes. It applies to projects with “mixed” experience levels within the team, and where the team members have some experience with related systems but are not fully familiar with the current project’s aspects²¹.
3. **Embedded Mode:** This mode is used for projects that are developed within a set of “tight” constraints. It is typically for software that is strongly coupled with complex hardware, or strict operational procedures. It requires a larger team and more experienced developers due to the complexity and creativity needed²¹.

Each mode uses different constants in the COCOMO equations to reflect the impact of these factors on the project’s cost and schedule. The choice of mode depends on the nature of the software project and the context in which it is being developed. The detailed COCOMO model also incorporates phases such as planning and requirements, system design, detailed design, module code and test, integration and test, and others to provide a comprehensive cost estimate¹.

Which feasibility study focuses on the cost aspect of the project ?Justify your answer with an example

The feasibility study that focuses on the cost aspect of a project is known as the **Financial Feasibility** study. This type of feasibility study assesses the financial viability of a proposed project by analyzing its expected costs and projected benefits in detail¹².

For example, consider a company planning to build a new manufacturing plant. A financial feasibility study would involve calculating the total capital investment required for the construction, machinery, labor, and other initial costs. It would also forecast the revenue the plant is expected to generate over time. The study would take into account various financial metrics such as the payback period, internal rate of return (IRR), and net present value (NPV) to determine if the project’s potential returns justify the investment².

The study would also consider the availability of financial resources, the cost of borrowing funds, and the impact on the company’s cash flow. If the financial

feasibility study concludes that the project is likely to be profitable and sustainable, it can be a green light for the company to proceed with the investment¹.

Distinguish between unit testing and integration testing

Unit testing and integration testing are both crucial in the software development process, but they serve different purposes and are conducted at different stages of the Software Development Life Cycle (SDLC). Here's how they differ:

Unit Testing:

- **Scope:** Involves testing individual components or units of the software to ensure each part functions correctly on its own¹.
- **Knowledge of Design:** The tester (usually the developer) knows the internal design of the software¹.
- **Order of Execution:** It is performed first, before integration testing¹.
- **Type of Testing:** Considered white box testing, as it requires knowledge of the code and internal structure¹.
- **Who Performs It:** Typically performed by the developer¹.
- **Defect Detection:** Easier to detect defects since it's focused on a small, isolated part of the system¹.
- **Cost:** Less costly due to its limited scope¹.
- **Dependencies:** Does not ensure the proper working of code with external dependencies¹.
- **Maintenance:** More cost-effective maintenance¹.
- **Execution Speed:** Faster execution compared to integration testing¹.

Integration Testing:

- **Scope:** Focuses on testing the interfaces between units/modules to ensure they work together as intended¹.
- **Knowledge of Design:** Does not require knowledge of the internal design of the software¹.
- **Order of Execution:** Performed after unit testing and before system testing¹.
- **Type of Testing:** Considered black box testing, as it tests the interfaces and interactions without needing to know the internal workings¹.
- **Who Performs It:** Usually performed by testers¹.
- **Defect Detection:** More difficult to detect defects since it involves multiple components working together¹.
- **Cost:** More costly due to the complexity of testing multiple integrated units¹.

- **Dependencies:** Ensures the proper working of code with external dependencies¹.
- **Maintenance:** Maintenance can be more expensive¹.
- **Execution Speed:** Slower because it involves the integration of multiple modules¹.

In summary, **unit testing** is about verifying that each piece of the software performs as designed in isolation, while **integration testing** ensures that different pieces of the software work together correctly when combined.

Which is more advisable to choose - High cohesion of module or high coupling withing the modules for software designing building ?

In software design and building, it is more advisable to choose **high cohesion** within modules and **low coupling** between modules. This principle leads to a better-structured and more maintainable system. Here's why:

High Cohesion:

- High cohesion within a module means that the module's internal elements are closely related and focused on a single task or purpose¹.
- It enhances the module's functional strength and makes it easier to maintain, understand, and reuse².
- Modules with high cohesion are less likely to be affected by changes in other parts of the system, which improves the stability of the software³.

Low Coupling:

- Low coupling between modules indicates that each module operates independently of the others¹.
- This independence reduces the risk that changes in one module will require changes in others, making the system more flexible and easier to modify².
- It also simplifies testing, as modules can be tested in isolation without the need for complex integration testing⁴.

In summary, aiming for high cohesion within modules and low coupling between them is a key design principle that contributes to creating a robust, maintainable, and scalable software system⁵.

Why to select RMMM for effective planning and strategy?

Selecting the Risk Mitigation, Monitoring, and Management (RMMM) plan for effective planning and strategy is beneficial for several reasons:

1. **Risk Identification:** RMMM helps in identifying potential risks early in the project, which allows for more effective planning and risk management¹.
2. **Risk Mitigation:** The plan includes strategies to mitigate identified risks, reducing the likelihood of their occurrence and minimizing their impact on the project¹.
3. **Risk Monitoring:** Continuous monitoring ensures that new risks are identified promptly and that mitigation strategies are effectively implemented¹.
4. **Risk Management:** In case a risk materializes, the RMMM plan provides a clear approach to manage and address the issue, ensuring that the project remains on track¹.
5. **Cost-Effectiveness:** By addressing risks early, the RMMM plan can save costs that would otherwise be incurred if risks were not managed properly¹.
6. **Project Continuity:** It ensures that the project can continue smoothly even when risks become a reality, by having predefined responses to potential problems¹.
7. **Documentation:** RMMM provides a structured way to document all risk-related activities, making it easier to track and manage risks throughout the project lifecycle¹.
8. **Communication:** It facilitates better communication among team members and stakeholders about the risks and the measures taken to address them¹.
9. **Quality Assurance:** By focusing on risk management, the RMMM plan contributes to the overall quality of the project, as risks that could compromise quality are addressed proactively¹.

In summary, the RMMM plan is a comprehensive approach to risk management that supports effective planning and strategy by proactively addressing potential risks throughout the project lifecycle¹.

Which is more advisable to choose - High cohesion of module or high coupling withing the modules for software designing building ?

In software design and building, it is more advisable to choose **high cohesion** within modules and **low coupling** between modules. This principle leads to a better-structured and more maintainable system. Here's why:

High Cohesion:

- **High cohesion within a module means that the module's internal elements are closely related and focused on a single task or purpose¹.**
- **It enhances the module's functional strength and makes it easier to maintain, understand, and reuse².**

- Modules with high cohesion are less likely to be affected by changes in other parts of the system, which improves the stability of the software³.

Low Coupling:

- Low coupling between modules indicates that each module operates independently of the others¹.
- This independence reduces the risk that changes in one module will require changes in others, making the system more flexible and easier to modify².
- It also simplifies testing, as modules can be tested in isolation without the need for complex integration testing⁴.

In summary, aiming for high cohesion within modules and low coupling between them is a key design principle that contributes to creating a robust, maintainable, and scalable software system⁵.

ESE MAY 2022

Differentiate between Epic, User stories in Agile methodology with an appropriate Example.

In Agile methodology, **Epics** and **User Stories** represent different levels of detail and scope in the planning and execution of work. Here's how they differ, with an example for each:

Epics:

- **Definition:** Epics are large bodies of work that are broken down into smaller tasks, which are often too large to be completed in a single sprint¹.
- **Scope:** They capture a significant feature or process that may span multiple sprints or even release cycles².
- **Detail:** Epics are less detailed than user stories and often require further refinement to become actionable tasks¹.
- **Example:** An Epic could be “Implement a new checkout process for the online store,” which is a broad goal encompassing several features and improvements.

User Stories:

- **Definition:** User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system¹.
- **Scope:** They are narrowly focused on delivering a single functionality and are small enough to be completed within a single sprint³.

- **Detail:** User stories are more detailed and specific compared to epics, often including acceptance criteria¹.
- **Example:** A User Story under the above Epic could be “As an online shopper, I want to be able to save my payment details so that I can check out more quickly.”

To illustrate, if the Epic is “Implement a new checkout process,” a related User Story might be “As a returning customer, I want to use a one-click checkout option to make my purchase faster.” The Epic provides a broad objective, while the User Story defines a specific feature that contributes to achieving that objective.

Multiple User Stories will collectively fulfill the Epic².

Construct a Use Case Diagram for the following case study:-The Course ware management system that can be used to manage courses and classes for an organization that specialize in providing training. The organization offers a variety of courses in a variety of areas such as learning management techniques & understanding different software languages and technologies. Each course is made up of a set of topics. Tutors in the organization are assigned courses to teach according to the area that they are specialized in and their availability. The organization publishes and maintains a calendar of different courses and assigns tutors every year. There is a group of course administrators in the organization to manage the courses including course content, assign courses to the tutor and define the course schedule. The training organization aims to use the courseware management system to get better control and visibility to the management of courses as also to streamline the process of generating and managing the schedule of the different courses.(specification must).

Elaborate any one design pattern that provides a strategy to solve a common problem in software. Assume a suitable example to justify your answer.

One common design pattern that provides a strategy to solve a recurring problem in software development is the **Singleton Pattern**. This pattern ensures that a class has only one instance and provides a global point of access to that instance¹.

Problem: Often in software design, it's necessary to have exactly one instance of a class. For example, a system might need a single global configuration manager or a single object that coordinates actions across a system.

Solution: The Singleton Pattern solves this problem by:

- Making the default constructor private, to prevent other objects from using the `new` operator with the Singleton class.
- Creating a static method that acts as a constructor. This method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

If no instance of the Singleton class exists, the static method creates one; if an instance already exists, the method returns a reference to that object.

Example: Consider a logging class used in an application to log messages to a file. It's important that this class is a singleton to avoid multiple instances writing to the same log file, which could lead to file corruption or performance issues.

```
public class Logger {  
    private static Logger instance;  
  
    private Logger() {  
        // Private constructor to prevent instantiation.  
    }  
  
    public static Logger getInstance() {  
        if (instance == null) {  
            instance = new Logger();  
        }  
        return instance;  
    }  
  
    public void log(String message) {  
        // Log message to the log file.  
    }  
}  
  
// Usage:  
Logger logger = Logger.getInstance();  
logger.log("This is a log message.");
```

In this example, the `Logger` class has a private constructor and a public `getInstance` method. The `getInstance` method ensures that only one `Logger` instance is created and provides a global access point to that instance. Any attempt to log messages through the `Logger` class will use the same instance, ensuring that all messages are written to the same log file in a coordinated manner¹.

Consider a team is building an ecommerce website, the final target product has Search, Product information, a Shopping

basket, checkout, favorites, and customer reviews. Analyze the requirement and Compare how the team will develop the Product using Incremental Approach and Iterative approach.

When building an e-commerce website with features like Search, Product information, a Shopping basket, checkout, favorites, and customer reviews, the development approach can significantly impact the project's progress and outcome. Let's compare how a team would develop the product using the Incremental Approach and the Iterative Approach:

Incremental Approach:

- The development is divided into fully working slices, known as increments.
- Each increment adds a new feature or functionality to the final product.
- The first increment might include basic functionalities such as Search, Product information, adding products to a Shopping basket, and checkout.
This increment is only released once it's complete¹.
- Subsequent increments build upon the existing functionality. For example, the second increment could add the favorites feature, and the third increment could introduce customer reviews¹.
- Each new increment is released after it's fully developed and tested, ensuring that every release is a functional extension of the product.

Iterative Approach:

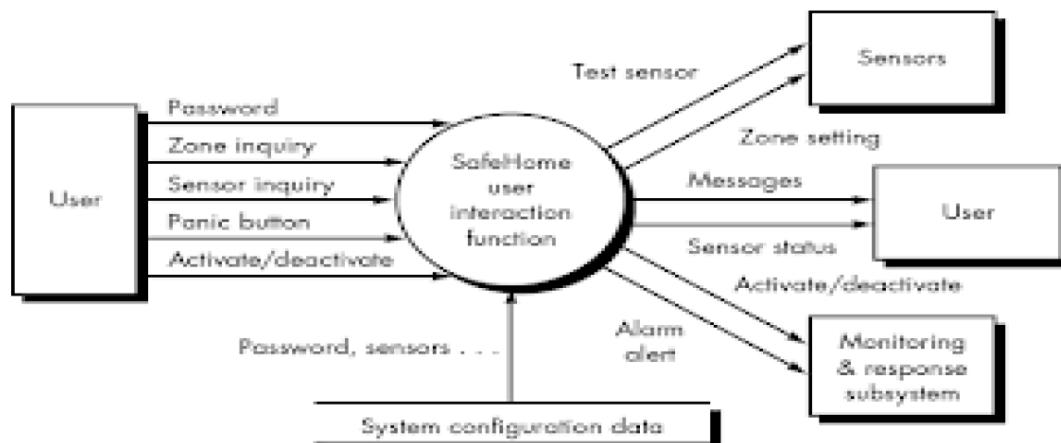
- The software is developed in successive cycles, each building on the previous one.
- A basic version of the entire product is released first, including all the required functionalities in a stripped-back form¹.
- Subsequent iterations improve existing features based on stakeholder feedback, analytics, and other inputs¹.
- The team continuously refines the product, adding new ideas and requirements or removing low-value areas in each iteration.
- This approach allows for frequent feedback and adaptation, making it suitable for projects with evolving requirements.

In the context of the e-commerce website:

- **Incremental:** The team would start by releasing a fully functional version of the core features. Once stable, they would incrementally add favorites and customer reviews in subsequent releases.
- **Iterative:** The team would release a basic version of all features initially. They would then iteratively enhance each feature, such as improving the

search algorithm or the user interface for the shopping basket, based on user feedback.

The choice between Incremental and Iterative approaches depends on factors like project requirements, team expertise, customer feedback mechanisms, and the desired speed of delivery. [Combining both approaches can also be beneficial, allowing for early user feedback through limited ‘Beta’ iterative releases before a full increment is released to all users¹.](#)



Consider the above diagram and Solve the function point for the project Assume all complexity adjustment factors & weighting factors are simple. $F_i = 46 \text{ FP} = 50 [0.65 + (0.01 \cdot 46)] = 56$

Measurement parameter	Count	Weighting Factor			=	FP
		Simple	Average	Complex		
Number of user inputs	3	3	4	6	=	9
Number of user outputs	2	4	5	7	=	8
Number of user inquiries	2	3	4	6	=	6
Number of files	1	7	10	15	=	7
Number of external interfaces	4	5	7	10	=	20
Count total					→	50

What are the different categories of Feasibility study which are carried in Software Requirement Specification?

In the context of Software Requirement Specification (SRS), a feasibility study is an assessment that aims to determine whether the proposed software project is likely to succeed. The feasibility study typically covers several key areas:

1. **Technical Feasibility:** This assesses whether the current technology and resources (both hardware and software) are adequate for developing the project. It also evaluates the technical team's skills and the ease of maintenance and upgrade for the chosen technology¹.
2. **Operational Feasibility:** This examines how well the software will operate in the live environment and how easy it will be to use and maintain after deployment. It also determines the usability of the product and whether the proposed solutions are acceptable¹.
3. **Economic Feasibility:** Also known as cost/benefit analysis, this study analyzes the costs associated with the project development, including hardware, software, design, development, and operational costs, and compares them to the financial benefits the project is expected to bring to the organization¹².
4. **Legal Feasibility:** This study looks at the legal aspects of the project, including data protection laws, social media laws, project certification, licenses, copyrights, and whether the proposed project conforms to legal and ethical requirements¹.
5. **Schedule Feasibility:** This area focuses on the project timelines and deadlines, analyzing how much time the team will need to complete the project and the impact of the project's timing on the organization¹.

These categories help stakeholders understand the various aspects of a project's viability and make informed decisions about whether to proceed with the development¹. Each type of feasibility study contributes a unique perspective and is crucial for the overall decision-making process in software engineering.

Which approach of Agile estimation techniques would you make use of if the user story needs to be prioritized in a short time?

For prioritizing user stories in a short time, the **MoSCoW method** is a practical and straightforward approach. It categorizes user stories into four priority levels:

- **Must Have:** Essential functionalities without which the product cannot be considered complete.
- **Should Have:** Important but not necessary for launch; can be included in future releases.
- **Could Have:** Desirable features that are not critical and can be included if time and resources permit.
- **Won't Have:** Features that are least critical or low value for the current iteration¹.

Another quick method is the **numerical (1-3) prioritization method**, where:

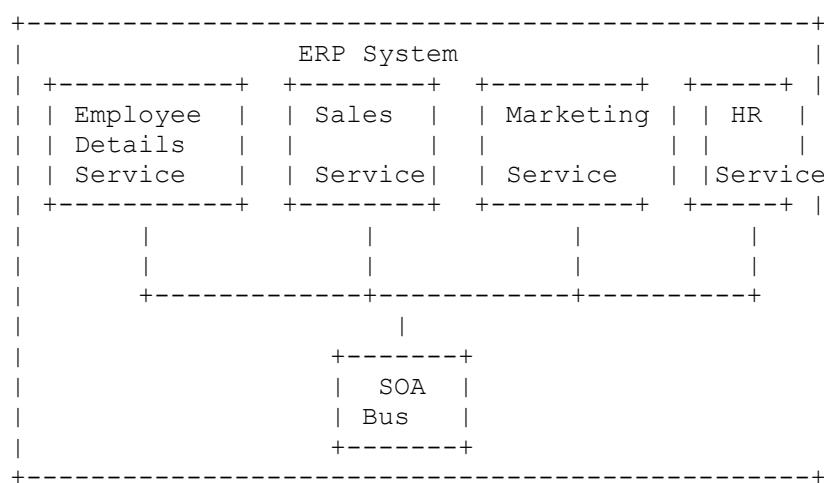
- Stories classed as **1s** are critical and must be delivered.
- Stories classed as **2s** have value but can be cut if necessary due to time or budget constraints.
- [Stories classed as 3s are lower priority or need further discussion¹.](#)

These methods help in making swift decisions on what to build first by understanding which stories are of the highest value and critical to success¹. They are particularly useful when time is limited and you need to focus on delivering the most impactful features first.

Consider an Enterprise Resource Planning System which has different modules Employee details, Sales, Marketing, Human Resource, etc... Which are interdependent in nature. Construct a suitable software architecture design with justification to show the interaction between them.

For an Enterprise Resource Planning (ERP) system with interdependent modules like Employee Details, Sales, Marketing, Human Resource, etc., a suitable software architecture design would be a **Service-Oriented Architecture (SOA)**. This architecture style enables services to communicate with each other and share data and processes through a network.

Here's a high-level design using SOA:



Justification:

- **Modularity:** Each module (Employee Details, Sales, Marketing, HR) is encapsulated as a service, which promotes modularity and makes the system easier to manage and maintain.

- **Interoperability:** SOA allows different services to communicate with each other despite being built in different programming languages or running on different platforms.
- **Reusability:** Common functionalities like authentication, logging, or data access can be centralized and reused across different services.
- **Scalability:** Services can be scaled independently, allowing the system to handle increased load by scaling only the necessary components.
- **Flexibility:** New services can be added or existing ones can be modified without affecting the rest of the system, as long as they adhere to the defined communication protocols.
- **Maintenance:** Issues can be isolated and addressed in individual services without impacting the entire system.

In this design, the **SOA Bus** acts as the communication layer that connects all services. It handles message routing, choreography, service orchestration, and provides the necessary middleware services required for communication between different modules.

Each module, such as Employee Details, Sales, Marketing, and HR, exposes its functionalities as a set of services that can be consumed by other modules. For example, the Sales module might need access to Employee Details for sales representative information, or the HR module might need to interact with the Sales module to process commissions.

By using SOA, the ERP system becomes more agile and better equipped to adapt to changing business needs. It also facilitates better data integration and workflow automation across different business processes.

Choose and explain appropriate Testing Technique ,Black Box or White Box is more suitable for following situations and justify your answer with help of suitable test cases (at least 5):

- 1.A user logged in when inputs a present username and correct password**
- 2.A user receives an error message when enters username and Incorrect password.**

For the given scenarios, **Black Box Testing** is more suitable because it focuses on testing the software functionality without needing to understand the internal code structure. Black Box Testing allows us to create test cases based on the requirements and specifications of the software, which is ideal for testing user login functionality.

Here are five test cases for each scenario:

Scenario 1: A user logged in when inputs a present username and correct password

1. **Valid Credentials:** Enter a valid username and the correct password. Expect successful login.
2. **Case Sensitivity:** Enter a valid username with different case and the correct password. Verify if the system is case-sensitive.
3. **Leading and Trailing Spaces:** Enter a valid username with leading/trailing spaces and the correct password. Expect login to fail or spaces to be trimmed.
4. **Password Masking:** Ensure the password is masked (shown as dots or asterisks) when entered.
5. **Session Management:** After successful login, navigate away from the application and return to check if the session persists.

Scenario 2: A user receives an error message when enters username and Incorrect password

1. **Incorrect Password:** Enter a valid username and an incorrect password. Expect an error message indicating incorrect credentials.
2. **Multiple Attempts:** Enter a valid username and incorrect password multiple times. Check for account lockout or captcha after a certain number of failed attempts.
3. **Error Message Clarity:** Verify that the error message does not specify whether the username or password is incorrect, to prevent information leakage.
4. **Copy-Paste Disabled:** Check if the password field disables copy-paste functionality for security reasons.
5. **Password Reset Link:** After a failed login attempt, ensure there is an option or link to reset the password.

These test cases cover various aspects of the login process, ensuring not only the functionality but also the security and user experience are up to standard. Black Box Testing is effective here as it simulates the actions of an end-user and verifies the system's behavior against expected outcomes.

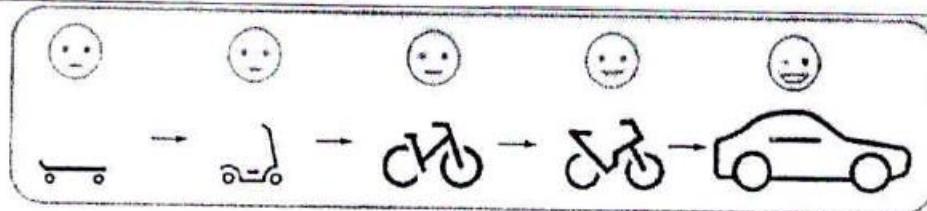
Google has spent several years developing self-driving cars which rely on a range of sensors, software and stored data and are now claimed to have covered hundreds of thousands of miles with a good safety record. A Google executive has now asked to meet the Transport Secretary, who has in turn asked your company for advice. Which Software Quality Factors will you select to ensure the product is up to the mark?

When advising on software quality factors for self-driving cars, it's crucial to focus on aspects that ensure the safety, reliability, and performance of the system. Here are the key software quality factors to consider:

1. **Reliability**: The software must perform its required functions under stated conditions for a specified period of time without failure. This is critical for self-driving cars, where system failures can have severe consequences¹.
2. **Safety**: The software should not cause any harm or hazard to the occupants or others in the environment. Compliance with safety standards like ISO 26262, which defines functional safety for automotive software, is essential¹.
3. **Maintainability**: The software should be easy to maintain and modify. Self-driving cars will require regular updates and improvements, and the software architecture should support this with minimal downtime¹.
4. **Performance**: The system must process inputs and react in real-time. Self-driving cars need to make split-second decisions based on sensor data, so high performance and low latency are non-negotiable¹.
5. **Testability**: Given the complexity of self-driving car software, it should be designed to facilitate thorough testing, including automated test cases, hardware-in-the-loop (HIL) testing, and mapping of test cases to requirements².
6. **Usability**: While self-driving cars aim to reduce the need for human intervention, the user interface should still be intuitive for monitoring and, if necessary, taking control of the vehicle¹.
7. **Scalability**: As self-driving technology evolves, the software must be able to scale with increasing data volumes and more complex algorithms without compromising performance¹.
8. **Security**: Protecting against cyber threats is vital. The software must include robust security measures to prevent unauthorized access and ensure data privacy¹.

These factors are interdependent and must be balanced to create a high-quality software product for self-driving cars. Ensuring these quality factors are met will contribute to a reliable, safe, and user-friendly self-driving car experience.

Q. 1



For above problem statement, Justify the Software Process Models you will prefer to use. (02 marks)

Write the general processing steps of the same. (04 Marks)

Compare Agile model w.r.t waterfall model on the parameter like measurement of progress, Customer interaction, Suitability, Test plan.

When comparing the Agile and Waterfall models, especially on parameters like measurement of progress, customer interaction, suitability, and test plan, here are the key differences:

Measurement of Progress:

- **Agile:** Progress is measured through the completion of iterations and the delivery of increments of the product. [The team frequently reviews the progress with stakeholders during sprint reviews or at the end of each iteration¹.](#)
- **Waterfall:** Progress is measured against a predefined project plan with distinct phases. [Each phase must be completed before moving on to the next, and progress is typically reviewed at the end of each phase².](#)

Customer Interaction:

- **Agile:** There is a high level of customer interaction as the customer is involved throughout the development process. [They provide feedback after each iteration, which is then incorporated into the next cycle¹.](#)
- **Waterfall:** Customer interaction is generally limited to the requirements gathering phase. [After this, the product is developed with little to no customer involvement until the final product is delivered².](#)

Suitability:

- **Agile:** Agile is suitable for projects where requirements are expected to change or are not fully understood at the outset. It's also well-suited for complex projects where quick delivery of parts of the project is beneficial¹.
- **Waterfall:** Waterfall is suitable for projects with well-defined requirements that are unlikely to change. It works well for smaller projects or where a strict sequence of steps is required, such as when compliance with regulations is necessary².

Test Plan:

- **Agile:** Testing is integrated throughout the development process. Tests are designed and executed in the same iteration as the development of the features they are testing¹.
- **Waterfall:** Testing is a distinct phase that occurs after the build phase. The test plan is typically developed early in the project but is executed after the product is fully developed².

In summary, Agile is a flexible, iterative approach that allows for frequent customer feedback and adapts to changes, whereas Waterfall is a structured, sequential approach that works best when requirements are clear and stable from the beginning.

How do you decide whether to conduct a software project using agile or waterfall methodology?

Deciding whether to use Agile or Waterfall methodology for a software project depends on various factors related to the project's requirements, complexity, and stakeholder involvement. Here are some key considerations to help you decide:

Project Size and Complexity:

- Agile is well-suited for projects with uncertain or evolving requirements¹. It allows for flexibility and adaptability as the project progresses.
- Waterfall is more appropriate for projects with well-defined requirements and where changes are not expected once the project starts¹.

Stakeholder Involvement:

- Agile requires close collaboration with stakeholders, with frequent communication and iterative feedback².
- Waterfall works well when stakeholder involvement is limited to key milestones and there is no need for continuous input².

Time to Market:

- Agile can provide incremental releases and faster delivery of features, which is beneficial if time to market is critical¹.
- Waterfall may lead to a longer time to market due to its sequential phases¹.

Risk Management:

- Agile allows for early identification and resolution of issues, reducing risks associated with changing requirements¹.
- Waterfall can be riskier if there is a high likelihood of changes or if the initial requirements are not thoroughly understood¹.

Documentation and Planning:

- Agile focuses on working software over comprehensive documentation, which can be a drawback if documentation is a regulatory requirement¹.
- Waterfall provides detailed documentation and planning, which can be beneficial for projects that require thorough documentation for future maintenance or compliance¹.

Testing and Quality:

- Agile integrates testing throughout the development process, allowing for continuous quality improvement¹.
- Waterfall typically involves a dedicated testing phase after development, which may delay the discovery of defects¹.

In summary, Agile is generally better for projects that require flexibility, frequent stakeholder input, and have a high degree of uncertainty. Waterfall is suitable for projects with clear requirements, less need for stakeholder involvement throughout the project, and where a structured approach is preferred. Sometimes, a hybrid approach that combines elements of both methodologies may be the best solution³. It's important to assess the specific needs and constraints of your project to make an informed decision.

According to your understanding, what makes a good functional requirement?

A good functional requirement is one that is **clear, specific, and testable**. It should accurately describe what the system should do and provide enough detail to guide the development process. Here are some key characteristics that make a functional requirement effective:

1. **Clarity:** It should be clearly stated, leaving no room for ambiguity or misinterpretation.
2. **Completeness:** It should fully describe the functionality to be delivered, including any necessary data inputs and expected outputs.
3. **Consistency:** It should not conflict with other requirements and should be aligned with the overall system objectives.
4. **Feasibility:** It should be realistic and achievable within the project's constraints, such as time, budget, and technology.
5. **Relevance:** It should be necessary for the system's intended purpose and contribute to achieving the business goals.
6. **Testability:** It should be verifiable through defined criteria, allowing testers to determine if the requirement has been met.
7. **Traceability:** It should be uniquely identifiable and traceable throughout the project lifecycle, from inception through to implementation and maintenance.

By ensuring that each functional requirement meets these criteria, you can help set the foundation for a successful project that delivers value to its users and stakeholders.

Which one of the following is a functional requirement ?

- a) Maintainability**
- b) Portability**
- c) Robustness**
- d) None of the mentioned**

The correct answer is **d) None of the mentioned**.

Functional requirements specify what the system should do. They include descriptions of data to be entered into the system, operations performed on these data, and the output of these operations. Functional requirements are concerned with the functionality of the system to solve a specific problem or provide specific features.

On the other hand, **Maintainability**, **Portability**, and **Robustness** are non-functional requirements. Non-functional requirements define how the system performs a certain function, rather than the functions themselves. They are constraints on the services or functions offered by the system such as performance, security, maintainability, and reliability criteria.

- **Maintainability** refers to how easy it is to maintain the system (i.e., to fix bugs or make improvements).
- **Portability** refers to the ease with which the software can be transferred from one environment to another.
- **Robustness** refers to how well the software can operate under stress or tolerate unpredictable or invalid input.

Therefore, none of the options listed are functional requirements.

Consider a system where, a heat sensor detects an intrusion and alerts the security company." What kind of a requirement the system is providing?

- a) Functional**
- b) Non-Functional**
- c) Known Requirement**
- d) None of the mentioned**

The system described is providing a **Functional requirement**. Functional requirements specify particular results of a system. This includes:

- Descriptions of data to be entered into the system
- Descriptions of operations performed by each screen
- Descriptions of work-flows performed by the system
- Descriptions of system reports or other outputs
- Who can enter the data into the system
- How the system meets applicable regulatory requirements

In this case, the heat sensor detecting an intrusion and alerting the security company is a specific operation that the system must perform, hence it is a functional requirement. The correct answer is **a) Functional**.

Write in brief about Agile Backlog Structure w.r.t. its product backlog.

In Agile methodology, the backlog structure is a critical component that helps teams prioritize and manage work throughout the development process. The product backlog is at the heart of this structure and serves as a dynamic, prioritized list of features, enhancements, bug fixes, and other changes that need to be made to

the product. Here's a brief overview of the Agile backlog structure with respect to its product backlog:

Product Backlog:

- The product backlog is a prioritized list of work for the development team that is derived from the roadmap and its requirements¹.
- It includes a variety of items such as user stories, features, bug fixes, technical tasks, and knowledge acquisition.
- The most important items are shown at the top of the product backlog so the team knows what to deliver first¹.
- The product backlog is continually updated and refined through a process known as backlog grooming or refinement.

Backlog Grooming:

- Regular review and maintenance of the product backlog ensure that it remains aligned with the project goals and customer needs¹.
- During backlog grooming sessions, items may be added, removed, or reprioritized based on feedback, market changes, or project progress.

User Stories and Epics:

- User stories are small, actionable items that describe a feature from the perspective of the end-user¹.
- Epics are larger work items that group related user stories, representing a bigger feature or function.

Sprint Backlog:

- The sprint backlog is a subset of the product backlog and contains items that the team commits to completing during a sprint².
- It is created during sprint planning meetings, where the team selects items from the product backlog based on priority and capacity.

The Agile backlog structure, with the product backlog at its core, enables teams to stay organized, prioritize effectively, and adapt to changes quickly, ensuring that they are always working on the most valuable tasks for the product and the customers. It's a living document that evolves with the project and helps the team maintain focus on delivering high-quality software that meets user needs¹².

Can you give me some examples of bad functional requirements?

Certainly! Here are some examples of bad functional requirements, which are often too vague, ambiguous, or unrealistic:

1. “**The system must have good usability**”: This requirement is subjective and lacks specific criteria for what constitutes “good usability”. A better requirement would define usability in measurable terms, such as "A customer service rep should be able to enter 3 issues in less than 15 minutes"¹.
2. “**Response time should be less than X seconds**”: Without context, this requirement is meaningless. It should specify under which conditions and for which functionalities the response time applies, such as "The search functionality should return results in less than 2 seconds on a standardized desktop within the firewall"¹.
3. “**Round-the-clock availability**”: This requirement fails to consider maintenance windows or acceptable downtime. It should be more realistic, like "The system should be available 99.9% of the time, excluding scheduled maintenance periods"¹.
4. “**The system shall work just like the previous one, but on a new platform**”: This requirement assumes that the previous system’s design is suitable for the new platform, which may not be the case. It should be rephrased to specify which aspects of the previous system’s functionality are to be preserved¹.
5. “**The system has to be bug-free**”: It’s unrealistic to expect any complex system to be completely free of bugs. A better requirement would focus on critical functionality and acceptable quality levels, such as "Critical bugs must be resolved before release, and the system should not have more than X minor bugs in production"¹.
6. “**Reporting**”: This requirement is too broad and doesn’t specify what kind of reporting is needed. It should detail the types of reports, their content, format, and frequency, like "The system must generate daily sales reports in PDF format"¹.
7. “**Make it accessible**”: Accessibility is important, but this requirement doesn’t specify to what standard or level. It should reference specific guidelines, such as "The system must meet WCAG 2.1 Level AA accessibility standards"¹.
8. “**X cannot change**”: This requirement is inflexible and doesn’t allow for necessary updates or improvements. It should allow for controlled changes with proper justification and approval processes¹.
9. “**Easy to use**”: Similar to “good usability”, this is subjective. It should be replaced with specific, user-centered design criteria and acceptance tests¹.
10. “**It has to be robust**”: The term “robust” is too general. The requirement should specify what robustness means in the context of the system, such as "The system must handle up to X concurrent users without performance degradation"¹.

These examples illustrate the importance of writing clear, specific, and measurable functional requirements to avoid ambiguity and ensure that the development team

has a solid understanding of what needs to be built. Good requirements are the foundation of successful software development projects.

Which one of the following is a requirement that fits in a developer's module?

a) Availability

b) Testability

c) Usability

d) Flexibility

The requirement that fits in a developer's module is b) Testability.

Testability refers to the ease with which software can be tested to ensure it functions correctly and meets its design specifications. It is a key consideration for developers when designing and writing code, as it impacts the effectiveness and efficiency of the testing process. Features that enhance testability include clear and concise code, modular design, and the use of testing frameworks and tools. By focusing on testability, developers can create software that is easier to debug, validate, and maintain.

How Does FDD Work? Explain with an Example

Feature-Driven Development (FDD) is an iterative and incremental software development process that is customer-centric and focuses on delivering tangible software results efficiently. [It follows a five-step development cycle centered around feature development¹²³](#). Here's how FDD works, explained with an example:

1. Develop an Overall Model: The team creates an overall model that serves as a high-level walkthrough of the system's scope and context. This model is developed through domain walkthroughs and discussions with subject matter experts.

Example: For a banking application, the team would create a model outlining the main modules like account management, transaction processing, and customer support.

2. Build a Features List: The team lists out all the features that the system must have. A feature in FDD is a small, client-valued function expressed in the form "action" (verb) "result" (noun).

Example: “Transfer funds between accounts” or “Generate monthly account statements”.

3. Plan by Feature: The features are grouped by their dependencies and planned for development. This step involves creating a high-level plan for the features.

Example: Prioritizing features that must be developed first, such as user authentication before fund transfer.

4. Design by Feature: For each feature, detailed design sessions are held, and design packages are created.

Example: Designing the user interface and backend services for the “Transfer funds” feature.

5. Build by Feature: The features are built in short iterations. Each feature is coded, tested, and integrated into the main build.

Example: Developing the “Transfer funds” feature, testing it thoroughly, and then integrating it with the banking application.

FDD emphasizes regular builds, status reporting, and feature delivery, which helps in tracking progress and ensuring that the development aligns with customer needs⁴. It is particularly useful for larger teams and projects where structuring development around specific, client-valued features can lead to more efficient and predictable results.

ESE MARCH 2023

What ideas validate the formation of Agile Principles and Values ?

The Agile Principles and Values were formed based on several key ideas that aimed to address the shortcomings of traditional software development methodologies, like the Waterfall model. These ideas include:

1. **Customer Satisfaction:** Agile prioritizes customer satisfaction through early and continuous delivery of valuable software. This idea is rooted in the belief that customers' needs and requirements can change over time, and software development should be flexible enough to accommodate these changes¹².

2. **Embrace Change:** Agile methodologies embrace change, even late in the development process. This flexibility allows teams to adapt and make changes that can lead to a better final product¹.
3. **Incremental Delivery:** Agile supports the notion of delivering working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale. This allows stakeholders to see progress and provides opportunities to adjust the product as needed².
4. **Collaboration:** Agile emphasizes collaboration between the business stakeholders and developers throughout the project. Regular communication is key to understanding the project's direction and ensuring that the team is building the right product¹.
5. **Motivated Individuals:** Agile recognizes that motivated individuals are the most important factor in project success. Providing them with the environment and support they need, and trusting them to get the job done, is essential¹.
6. **Face-to-Face Communication:** Agile methodologies value face-to-face communication as the most efficient and effective method of conveying information to and within a development team¹.
7. **Working Software:** In Agile, working software is the primary measure of progress. This focus on producing functional software at every step ensures that the product is always in a potentially shippable state¹.
8. **Sustainable Development:** Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely¹.
9. **Continuous Attention to Technical Excellence:** Agile processes enhance agility through continuous attention to technical excellence and good design¹.
10. **Simplicity:** Agile promotes simplicity—the art of maximizing the amount of work not done—is essential¹.
11. **Self-Organizing Teams:** Agile methodologies suggest that the best architectures, requirements, and designs emerge from self-organizing teams¹.
12. **Reflect and Adjust:** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly¹.

These ideas were a response to the frustration with the slow and rigid nature of traditional software development practices. The Agile Manifesto and its principles were formalized by a group of software practitioners who sought a more rational, human approach to software development, one that could respond quickly to changing requirements and improve the overall quality and speed of software delivery¹².

Justify why one of the agile estimation techniques is considered to be more of a decision making tool

One of the Agile estimation techniques that is considered to be more of a decision-making tool is **Planning Poker**. Planning Poker, also known as Scrum Poker, is an estimation technique that combines expert opinion, analogy, and disaggregation into an enjoyable approach to estimating, usually to size user stories in Scrum¹.

Here's why Planning Poker is seen as a decision-making tool:

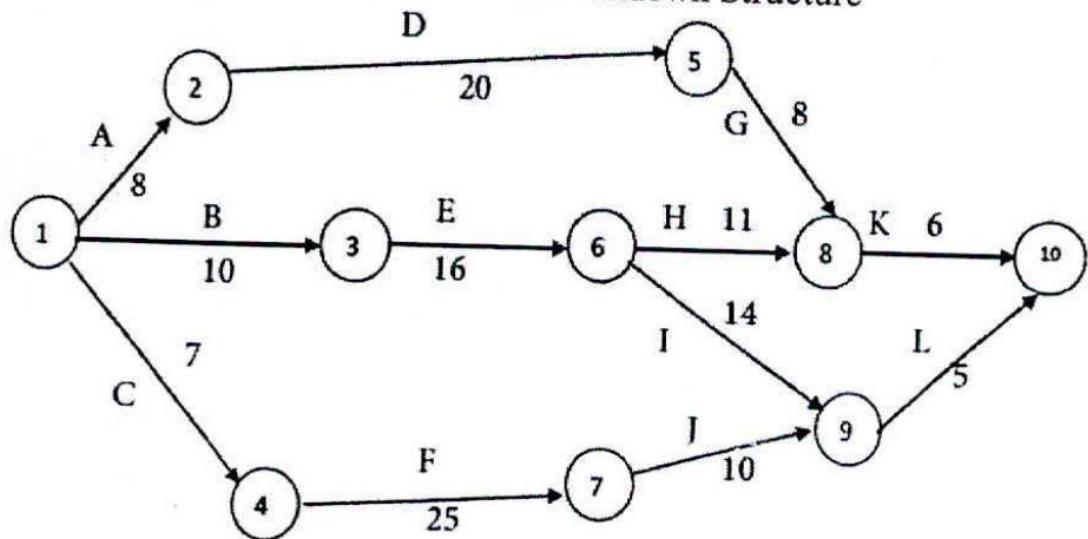
1. **Collaborative Estimation:** It involves the entire team, which means that every member's perspective and expertise are considered in the estimation process. This collaborative effort leads to more accurate and democratic decision-making¹.
2. **Consensus Building:** Team members discuss their estimates, and through this discussion, they reach a consensus. This process ensures that the team collectively agrees on the effort required, making it a group decision rather than an individual one¹.
3. **Facilitates Discussion:** Discrepancies in estimates prompt discussions, uncovering aspects of the work that might not have been considered initially. This helps the team make informed decisions about the complexity and effort required for a user story¹.
4. **De-emphasizes Anchoring:** By having team members reveal their estimates simultaneously, Planning Poker reduces the influence of the first estimate on subsequent ones, leading to more independent and unbiased decision-making¹.
5. **Improves Understanding:** As team members explain their reasoning for their estimates, it enhances the understanding of the user story for the entire team, which is crucial for making sound decisions¹.

In summary, Planning Poker is not just about estimating effort; it's a tool that facilitates team engagement, discussion, and consensus, which are all essential elements of effective decision-making in Agile project management¹.

You are appointed as a Project Manager of ABC Multimedia Ltd, the project allotted is of 8 months duration. The budget of the Project is around 85 lacs. You have the responsibility to create the following (2mks each)

- a) OBS b) WBS c) R.A.M d) Staff Management Plan

Following is the given data of WorkBreakdown Structure



- Calculate ES,LS,EF,LF for each node
- Determine the Critical Path and Critical Activities
- Explain what do you mean by Critical Path

Construct a suitable software architecture diagram to represent the interaction for the following scenario (Any 1) -

- (a) In video processing, an unprocessed video undergoes a series of transformations to become a processed video that can serve a more useful purpose. The image frames from the live video recorded serve as the input data and are sent into the application via the data source. Once in the pipeline, the data is transported via pipes between each component. From the data source, the data goes through a series of filters sequentially, each processing the data to make it more useful for the next filter order to achieve the eventual goal of object detection. Eventually, the processed data, which in this case is the input image frame with bounding boxes drawn around objects of interest, is served as the application's output in the data sink.

(b) Let's say your directory/ folder for this semester contains many files from different subjects and is extremely unorganised. You wish to create a SE folder which should have SE lecture slides.

Let's also assume your lecture slides should be organised chapterwise where each chapter is a folder and saved by respective module name e.g. "SE Unit 1 Introduction Topic".

Categorize and Explain at least two types of Requirement

Elicitation techniques which are used for developing Long term Projects.

For developing long-term projects, requirement elicitation techniques are crucial to understand and document the needs and expectations of stakeholders. Here are two types of requirement elicitation techniques that are particularly useful for long-term projects:

1. Interviews:

- **Category:** Direct Elicitation Technique
- **Description:** Interviews involve one-on-one or group discussions with stakeholders to gather detailed information about their requirements. They can be structured, with a predefined set of questions, or unstructured, allowing for open-ended discussions¹.
- **Long-term Project Suitability:** Interviews are valuable for long-term projects as they provide an in-depth understanding of stakeholder needs, which may evolve over time. They allow for immediate clarification of doubts and can uncover implicit requirements that stakeholders may not initially express¹.

2. Brainstorming Sessions:

- **Category:** Collaborative Elicitation Technique
- **Description:** Brainstorming sessions are group activities that encourage the free flow of ideas and creative thinking among stakeholders. A facilitator typically guides the session to ensure that every participant has the opportunity to contribute¹.

- **Long-term Project Suitability:** Brainstorming is effective for long-term projects because it can generate a wide range of ideas and solutions, some of which may only become relevant as the project progresses. It also fosters a sense of collaboration and buy-in from stakeholders, which is essential for the success of long-term initiatives¹.

These techniques help in creating a comprehensive and well-defined set of requirements that serve as the foundation for the design and development of long-term projects. They ensure that the project is aligned with stakeholder expectations and is adaptable to future changes¹.

Following password field accepts minimum 6 characters and maximum

10 characters.

Enter Password: _____

a) Apply Equivalence Partitioning on the given scenario.

b) Apply Boundary Value Analysis on the given scenario

Equivalence Partitioning and Boundary Value Analysis are two black box testing techniques used to design test cases. Here's how they can be applied to the given scenario:

a) Equivalence Partitioning (EP): Equivalence Partitioning involves dividing input data into partitions of equivalent data from which test cases can be derived. For the password field scenario, the input data can be divided into three partitions:

1. **Invalid Partition (Less than 6 characters):** Passwords with 0 to 5 characters are invalid and should not be accepted.
2. **Valid Partition (6 to 10 characters):** Passwords with 6 to 10 characters are valid and should be accepted.
3. **Invalid Partition (More than 10 characters):** Passwords with more than 10 characters are invalid and should not be accepted.

Test cases from each partition:

- Test Case 1: Enter 5 characters (e.g., “abcde”) - Expect rejection.
- Test Case 2: Enter 6 characters (e.g., “abcdef”) - Expect acceptance.
- Test Case 3: Enter 10 characters (e.g., “ABCDEFGHIJ”) - Expect acceptance.
- Test Case 4: Enter 11 characters (e.g., “ABCDEFGHIJK”) - Expect rejection.

b) Boundary Value Analysis (BVA): Boundary Value Analysis involves designing test cases that include values at the boundaries. For the password field scenario, the boundaries are at 6 and 10 characters:

1. **Just below the valid boundary:** 5 characters (invalid).
2. **At the lower valid boundary:** 6 characters (valid).
3. **At the upper valid boundary:** 10 characters (valid).
4. **Just above the valid boundary:** 11 characters (invalid).

Test cases at each boundary:

- Test Case 1: Enter 5 characters (e.g., “abcde”) - Expect rejection.
- Test Case 2: Enter 6 characters (e.g., “abcdef”) - Expect acceptance.
- Test Case 3: Enter 10 characters (e.g., “abcdefghij”) - Expect acceptance.
- Test Case 4: Enter 11 characters (e.g., “abcdefghijk”) - Expect rejection.

Additionally, it's good practice to test the exact boundary values:

- Test Case 5: Enter exactly 6 characters (e.g., “abcdef”) - Expect acceptance.
- Test Case 6: Enter exactly 10 characters (e.g., “abcdefghij”) - Expect acceptance.

These techniques help ensure that the password field behaves correctly at and around the defined limits, catching potential boundary-related defects.

Identify different types of Software Project Risk the Team encounters

in the complete Project Life cycle.

During the complete project lifecycle, a software development team may encounter various types of risks. These risks can be broadly classified into three main categories:

1. **Project Risks:** These are related to budgetary, schedule, personnel, resource, and customer-related problems. [A significant project risk is schedule slippage, which can affect the project development and delivery¹.](#)
2. **Technical Risks:** These involve potential method, implementation, interfacing, testing, and maintenance issues. [Technical risks also include ambiguous or incomplete specifications, technical uncertainty, and technical obsolescence².](#)
3. **Business Risks:** These risks are associated with the marketability of the software product, such as building an excellent product that no one needs, losing budgetary or personnel commitments, and other similar concerns².

Other risk categories include:

- **Known Risks:** Risks that can be identified after careful assessment of the project plan and the business and technical environment in which the project is being developed².
- **Predictable Risks:** Risks that are hypothesized from previous project experience².
- **Unpredictable Risks:** Risks that can and do occur but are extremely difficult to identify in advance².

Effective risk management involves identifying, addressing, and eliminating these risks before they can damage the project. It's crucial for the project team to continuously monitor and manage these risks throughout the project lifecycle to ensure the success of the software project.

Using what you have learned ,how would you solve the real time challenges in Software Engineering (Select any 1 topic) :-

1 .Cohesion and Coupling

2.3R -Refactoring Reengineering Reusability

3. Cleanroom Software engineering

4. Cost Benefit Analysis

To address real-time challenges in Software Engineering, I would focus on the topic of **3R - Refactoring, Reengineering, and Reusability**. These three practices are crucial for maintaining and improving the quality of software systems, especially as they evolve and grow over time.

Refactoring is the process of restructuring existing code without changing its external behavior. It's essential for keeping the codebase clean, understandable, and maintainable. By regularly refactoring, you can improve the design of the software, make it easier to comprehend, reduce complexity, and find and fix hidden bugs¹.

Reengineering, also known as software renovation or reconstruction, involves examining and altering a system to reconstitute it in a new form. The goal is to retain and extend the value of the software system as it evolves. Reengineering can involve major restructuring or rewriting parts of the system to enhance its performance or maintainability².

Reusability is about designing software components in a way that allows them to be reused in different systems or in future projects. This reduces the overall development time and costs by leveraging existing, tested components rather than creating new ones from scratch².

Here's how I would apply these practices to solve real-time challenges:

1. **Implement Continuous Refactoring:** Integrate refactoring into the regular development cycle. Encourage developers to refactor as they work, rather than treating it as a separate task.
2. **Establish a Reengineering Strategy:** Assess the current state of the software to determine which parts would benefit most from reengineering. Develop a plan that prioritizes these areas and outlines the steps needed to reengineer them effectively.
3. **Promote Reusability:** Create a culture that values reusability. Encourage developers to write modular, well-documented code and to contribute to a shared repository of reusable components.
4. **Invest in Automated Testing:** Automated tests are crucial for refactoring and reengineering efforts. They provide a safety net that ensures changes do not introduce new bugs.
5. **Provide Training and Tools:** Equip the team with the knowledge and tools necessary to refactor, reengineer, and design for reusability. This might include training sessions, workshops, and access to software that supports these practices.

By focusing on Refactoring, Reengineering, and Reusability, software engineering teams can enhance the quality and longevity of their software products, making them more adaptable to changing requirements and technologies¹².

Categorize and Explain at least two types of Requirement

Elicitation techniques which are used for developing Long term Projects.

For developing long-term projects, requirement elicitation techniques are crucial to understand and document the needs and expectations of stakeholders. Here are two types of requirement elicitation techniques that are particularly useful for long-term projects:

1. **Interviews:**
 - o **Category:** Direct Elicitation Technique

- **Description:** Interviews involve one-on-one or group discussions with stakeholders to gather detailed information about their requirements. They can be structured, with a predefined set of questions, or unstructured, allowing for open-ended discussions¹.
- **Long-term Project Suitability:** Interviews are valuable for long-term projects as they provide an in-depth understanding of stakeholder needs, which may evolve over time. They allow for immediate clarification of doubts and can uncover implicit requirements that stakeholders may not initially express¹.

2. Brainstorming Sessions:

- **Category:** Collaborative Elicitation Technique
- **Description:** Brainstorming sessions are group activities that encourage the free flow of ideas and creative thinking among stakeholders. A facilitator typically guides the session to ensure that every participant has the opportunity to contribute¹.
- **Long-term Project Suitability:** Brainstorming is effective for long-term projects because it can generate a wide range of ideas and solutions, some of which may only become relevant as the project progresses. It also fosters a sense of collaboration and buy-in from stakeholders, which is essential for the success of long-term initiatives¹.

These techniques help in creating a comprehensive and well-defined set of requirements that serve as the foundation for the design and development of long-term projects. They ensure that the project is aligned with stakeholder expectations and is adaptable to future changes¹.

Given the following values, compute function points when all complexity adjustment factors (CAF) and weighting factors are average.

User Input = 40, User Output = 30, User Inquiries = 25, User Files = 5, External Interface = 3

To compute the function points when all complexity adjustment factors (CAF) and weighting factors are average, we can follow these steps:

1. **Determine the Complexity Adjustment Factor (CAF):** Since the complexity adjustment factor is average, we use a scale of 3. The formula for CAF is:

$$CAF=0.65+(0.01\times F)$$

where (F) is the sum of the degree of influence, which is (14 \times \text{scale}). For an average scale of 3:

$$F=14\times3=42$$

Therefore:

$$CAF=0.65+(0.01\times42)=0.65+0.42=1.07$$

2. **Calculate Unadjusted Function Points (UFP): We use the average weighting factors for each type of function point¹:**

- User Input (EI): 4
- User Output (EO): 5
- User Inquiries (EQ): 4
- User Files (ILF): 10
- External Interface (EIF): 7

Now, we multiply each count by its respective weighting factor:

$$UFP=(40\times4)+(30\times5)+(25\times4)+(5\times10)+(3\times7)$$

$$UFP=160+150+100+50+21$$

$$UFP=481$$

3. **Compute the Total Function Points (FP): Finally, we multiply the UFP by the CAF to get the total function points:**

$$FP=UFP\times CAF$$

$$FP=481\times1.07$$

$$FP=514.67$$

Therefore, the function points for the project, when all complexity adjustment factors and weighting factors are average, is **514.67**.

Identify different types of Software Project Risk the Team encounters in the complete Project Life cycle.

During the complete project lifecycle, a software development team may encounter various types of risks. These risks can be broadly classified into three main categories:

1. **Project Risks:** These are related to budgetary, schedule, personnel, resource, and customer-related problems. A significant project risk is schedule slippage, which can affect the project development and delivery¹.

2. **Technical Risks:** These involve potential method, implementation, interfacing, testing, and maintenance issues. Technical risks also include ambiguous or incomplete specifications, technical uncertainty, and technical obsolescence².
3. **Business Risks:** These risks are associated with the marketability of the software product, such as building an excellent product that no one needs, losing budgetary or personnel commitments, and other similar concerns².

Other risk categories include:

- **Known Risks:** Risks that can be identified after careful assessment of the project plan and the business and technical environment in which the project is being developed².
- **Predictable Risks:** Risks that are hypothesized from previous project experience².
- **Unpredictable Risks:** Risks that can and do occur but are extremely difficult to identify in advance².

Effective risk management involves identifying, addressing, and eliminating these risks before they can damage the project. It's crucial for the project team to continuously monitor and manage these risks throughout the project lifecycle to ensure the success of the software project.

Using what you have learned ,how would you solve the real time challenges in Software Engineering (Select any 1 topic) :-

1 .Cohesion and Coupling

2.3R -Refactoring Reengineering Reusability

3. Cleanroom Software engineering

4. Cost Benefit Analysis

To address real-time challenges in Software Engineering, I would focus on the topic of **3R - Refactoring, Reengineering, and Reusability**. These three practices are crucial for maintaining and improving the quality of software systems, especially as they evolve and grow over time.

Refactoring is the process of restructuring existing code without changing its external behavior. It's essential for keeping the codebase clean, understandable, and maintainable. By regularly refactoring, you can improve the design of the

software, make it easier to comprehend, reduce complexity, and find and fix hidden bugs¹.

Reengineering, also known as software renovation or reconstruction, involves examining and altering a system to reconstitute it in a new form. The goal is to retain and extend the value of the software system as it evolves. Reengineering can involve major restructuring or rewriting parts of the system to enhance its performance or maintainability².

Reusability is about designing software components in a way that allows them to be reused in different systems or in future projects. This reduces the overall development time and costs by leveraging existing, tested components rather than creating new ones from scratch².

Here's how I would apply these practices to solve real-time challenges:

1. **Implement Continuous Refactoring:** Integrate refactoring into the regular development cycle. Encourage developers to refactor as they work, rather than treating it as a separate task.
2. **Establish a Reengineering Strategy:** Assess the current state of the software to determine which parts would benefit most from reengineering. Develop a plan that prioritizes these areas and outlines the steps needed to reengineer them effectively.
3. **Promote Reusability:** Create a culture that values reusability. Encourage developers to write modular, well-documented code and to contribute to a shared repository of reusable components.
4. **Invest in Automated Testing:** Automated tests are crucial for refactoring and reengineering efforts. They provide a safety net that ensures changes do not introduce new bugs.
5. **Provide Training and Tools:** Equip the team with the knowledge and tools necessary to refactor, reengineer, and design for reusability. This might include training sessions, workshops, and access to software that supports these practices.

By focusing on Refactoring, Reengineering, and Reusability, software engineering teams can enhance the quality and longevity of their software products, making them more adaptable to changing requirements and technologies¹².

Apply appropriate Test cases for give User Story using Agile Testing

method for the given scenario -

User story for the shopping cart - As a buyer I want to be able to add items to my shopping cart So I can purchase them.

To apply appropriate test cases for the given user story using Agile Testing methods, we need to create scenarios that cover all the functionalities mentioned in the user story. Here are some test cases for the shopping cart feature:

1. Test Case: Add a Single Item to the Shopping Cart

- **Objective:** Verify that a buyer can add a single item to the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Navigate to the product page.
 3. Select a single item.
 4. Click the “Add to Cart” button.
- **Expected Result:** The item should be added to the shopping cart, and the cart should display one item.

2. Test Case: Add Multiple Items to the Shopping Cart

- **Objective:** Ensure that a buyer can add multiple items to the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Navigate to the product pages.
 3. Select multiple items.
 4. Click the “Add to Cart” button for each item.
- **Expected Result:** All selected items should be added to the shopping cart, and the cart should display the correct number of items.

3. Test Case: Add the Same Item Multiple Times

- **Objective:** Confirm that a buyer can add the same item multiple times to the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Navigate to the product page of an item.
 3. Add the same item to the cart multiple times.
- **Expected Result:** The shopping cart should reflect the correct quantity for the repeated item.

4. Test Case: View Shopping Cart Contents

- **Objective:** Verify that a buyer can view the contents of the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Add items to the shopping cart.

3. Navigate to the shopping cart page.
- **Expected Result:** The shopping cart page should display all items that have been added.

5. Test Case: Update Item Quantity in the Shopping Cart

- **Objective:** Check if a buyer can update the quantity of an item in the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Add an item to the shopping cart.
 3. Go to the shopping cart page.
 4. Update the quantity of the item.
- **Expected Result:** The item's quantity should be updated, and the cart should reflect the new quantity.

6. Test Case: Remove Item from the Shopping Cart

- **Objective:** Ensure that a buyer can remove an item from the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Add items to the shopping cart.
 3. Navigate to the shopping cart page.
 4. Remove an item from the cart.
- **Expected Result:** The item should be removed from the shopping cart, and the cart should update accordingly.

7. Test Case: Proceed to Checkout with Items in the Shopping Cart

- **Objective:** Verify that a buyer can proceed to checkout with items in the shopping cart.
- **Steps:**
 1. Log in as a buyer.
 2. Add items to the shopping cart.
 3. Navigate to the shopping cart page.
 4. Click the "Proceed to Checkout" button.
- **Expected Result:** The buyer should be taken to the checkout page with the items from the shopping cart.

These test cases cover various scenarios that a buyer might encounter while using the shopping cart feature. They ensure that the functionality works as expected and meets the acceptance criteria defined in the user story. Each test case should be executed, and any deviations from the expected results should be reported and addressed.

You work for a large “social networking” company which has recently

introduced a one-to-one chat mechanism, promising that they will never

censor conversations. Users are now reporting that their friends' computers are being compromised by malicious software. When users

click on links within messages sent by this malicious software, their

machine is also compromised, and spreads the infection still further. A

crisis meeting has decided that the chat software must be modified to

block this “worm” behavior. According to you, s manager of this project, which software quality factors failed to ensure its working?

As a manager of the project, the software quality factors that failed to ensure the working of the chat software in preventing the spread of malicious software, or “worm” behavior, include:

1. **Reliability:** The software should consistently perform its required functions under stated conditions without failure. [The presence of a worm indicates a failure in the system's ability to maintain reliable operations¹.](#)
2. **Integrity:** This factor deals with the security of the software system, preventing unauthorized access or modifications. [The worm's ability to compromise user machines suggests a lack of integrity in the chat software¹.](#)
3. **Correctness:** The software must produce the correct output and functionality. [In this case, the chat mechanism failed to correctly identify and block malicious links, leading to incorrect behavior¹.](#)
4. **Usability:** While not directly related to the worm, usability includes the ease with which users can protect themselves from threats. [If the software had better usability features for security, users might have been able to avoid clicking on malicious links¹.](#)
5. **Maintainability:** The ease with which the software can be modified to correct faults. [The need for a crisis meeting to decide on modifications suggests that the software was not easily maintainable to quickly adapt to such threats¹.](#)

6. **Testability:** The software should be designed to facilitate testing, which includes security testing. [It appears that the chat software was not adequately tested for security vulnerabilities that could allow worm behavior¹.](#)

To address these issues, the chat software must be modified to include better security measures, such as link scanning and user education on potential threats. Additionally, the software should be designed for easier maintenance and testing to quickly respond to future threats. [Implementing these changes will help restore the software's reliability, integrity, and correctness, and improve its maintainability and testability¹.](#)

ESE DECEMBER 2022

Q3a)	<p>Draw control flow graph for the given graph ? Calculate cyclomatic complexity using no. of nodes and edges. Can cyclomatic complexity become equal to zero? Justify</p> <p>(1) { int i, j, k; (2) for (i=0 ; i<=N ; i++) (3) p[i] = 1; (4) for (i=2 ; i<=N ; i++) (5) { (6) k = p[i]; j=1; (7) while (a[p[j-1]] > a[k] { (8) p[j] = p[j-1]; (9) j--; (10) } (11) p[j]=k; (12) }</p>
------	---

Suppose you are the project manager of a software project requiring the following activities

Task Number	Task	Duration	Dependent on Task
1	Specification	15	-
2	Design database	45	1
3	Design GUI	30	1
4	Code database	105	2
5	Code GUI part	45	3
6	Integrate and test	120	4 and 5
7	Write user manual	60	1

- a) Determine Earliest Start (ES), Earliest Finish time (EF), Latest start time (LS), Latest finish (LF) for every task
- b) Compute Slack time (ST) for each task.

Teenagers who run a lawn mowing(cutting) service tend to struggle with administration tasks such as writing quotes and collecting payments. Aruna, who runs a mowing service, currently spends approximately two hours extra each week, besides mowing, on administration. The system that will be developed is expected to save Aruna's time. Additionally, approximately 5% of existing customers are unsatisfied as, for example, their lawn was not mowed as originally arranged. A new system needs to be developed, that supports quotes, cash and card payments, reminders, scheduling, and maps for directions for a teenager's mowing service. You are to implement the system using database and APIs. Currently there are only few team members available to work on the project. Choose the best process model that has minimum cost and maximum flexibility for this system. Describe how the model is applied to the project. Justify why it is better than other models.

For a project that requires minimum cost and maximum flexibility, especially with a small team like Aruna's lawn mowing service, the Agile methodology would be the best process model¹. Agile is focused on incremental and iterative

development, which allows for flexibility in responding to changes and prioritizes customer satisfaction¹.

Here's how Agile can be applied to Aruna's project:

1. **Project Initiation:** Begin with a meeting to understand Aruna's needs and define the minimum viable product (MVP) that includes essential features like scheduling, payment processing, and reminders.
2. **Iteration Planning:** Break down the project into small, manageable iterations or sprints. Each sprint will focus on delivering a set of features from the MVP.
3. **Development and Testing:** In each sprint, the team develops and tests the features. Agile encourages continuous integration and testing, which helps in identifying issues early.
4. **Customer Feedback:** After each sprint, present the increment to Aruna for feedback. This ensures that the system meets her needs and allows for adjustments based on her input.
5. **Iterative Improvement:** Use the feedback to improve the system in subsequent sprints. This iterative process ensures that the system evolves to meet Aruna's changing requirements.

Agile is better than other models like Waterfall because it allows for more frequent reassessment and adaptation, which is crucial for a small, dynamic project¹. It also enables the team to deliver value early and often, which can help in satisfying the customers who are currently unsatisfied with the service. Moreover, Agile's collaborative approach ensures that the team can work effectively even with limited members, making it cost-effective and flexible for Aruna's needs.

What are the fundamental activities common to all process models?

The fundamental activities common to all software process models are:

1. **Software Specification:** Defining what the system should do, including the requirements and constraints of the software product¹.
2. **Software Development:** The actual production of the software to meet the specified requirements, which includes design and implementation¹.
3. **Software Validation:** Checking that the software does what the customer wants and that it fulfills the specified requirements¹.
4. **Software Evolution:** Adapting and evolving the software to meet changing customer needs and to address issues found after the deployment¹.

These activities are integral parts of the software development lifecycle and are present in various forms across different process models like Waterfall, Agile,

[Scrum](#), and [DevOps](#)²³. They ensure that the software is developed systematically, meets user requirements, and can adapt over time as needed.

Differentiate between incremental and iterative model

The **Incremental** and **Iterative** models are both approaches used in software development, but they differ in how the final product is constructed and delivered over time:

Incremental Model:

- [In the incremental development model, the product is designed, implemented, and tested incrementally \(a little more is added each time\) until the product is finished¹.](#)
- It involves both development and maintenance.
- The product is decomposed into a number of components, each of which is designed and built separately (termed as increments).
- Each component is delivered to the client when it is complete.
- This allows partial implementation of the product and determines the direction of the development process early on.
- The incremental model is flexible and less costly to change scope and requirements.

Iterative Model:

- [In the iterative development model, initial development produces a version of the new system, which is then refined through many versions until the final version is achieved¹.](#)
- It involves the repetition of the same development cycle.
- Each version is based on the previous one and incorporates further features and refinements.
- Feedback from the previous iteration is used to improve the next version.
- The iterative model is good for reducing the level of risk and allowing the project to adapt to changes quickly.

In summary, the **incremental model** focuses on delivering a working version of the product through gradual increases in functionality, while the **iterative model** focuses on refining the product through repeated cycles. [The incremental model is useful when the requirements are well understood and changes are expected to be minimal during the development process, whereas the iterative model is useful when the project is large and requirements are not well understood or are expected to change significantly¹.](#)

A local airport decides to build an Air Traffic Control(ATC) Assistant

system , a high risk project, to manage 20M aircraft in vicinity of airport. The ATC Assistant has several functions. Using the system it is

possible to track location of aircraft on ground or in air, and keep track

of who is controlling the aircraft. It provides a decision support system

for tracking and altering the states of aircraft in designated air traffic

zones. There is a module that is used to train air traffic controllers through ATC simulation. A display of weather sensors is also provided.

Aircrafts are identified whenever they enter a controlled air space either

through a TRACON. Aircraft leaving the control space are assigned to

an en route center and removed from the list.

Choose the best process model for this system. Describe how the model

is applied to the project. Justify why it is better than other models.

For the described Air Traffic Control (ATC) Assistant system, the best process model would be the **Spiral Model**. This model is particularly well-suited for high-

risk projects like an ATC system due to its focus on iterative development and risk management.

Application of the Spiral Model:

1. **Objective Setting:** Define the objectives for the iteration, including performance, functionality, and ability to handle 20M aircraft.
2. **Risk Assessment:** Identify risks such as system failure, inaccurate tracking, or inadequate training simulation, and develop strategies to mitigate them.
3. **Development and Testing:** Develop and test the system incrementally, starting with core functionalities like tracking aircraft and managing air traffic zones.
4. **Planning:** Review the progress, evaluate the results, and plan the next iteration, which could include enhancing the decision support system or integrating weather sensor data.

Justification:

- **Risk Management:** The Spiral Model emphasizes risk analysis and mitigation, which is crucial for an ATC system where safety is paramount¹.
- **Flexibility:** It allows for incorporating changes and new requirements, such as updates in air traffic regulations or technology advancements².
- **User Feedback:** Stakeholders, including air traffic controllers and airport management, can provide feedback early and throughout the development process.
- **Incremental Releases:** The system can be developed in increments, ensuring that the most critical features are prioritized and tested thoroughly.

Compared to other models like the Waterfall or V-Model, which are more rigid and less adaptable to change, the Spiral Model offers a more flexible and risk-averse approach. It's better suited for complex, high-risk projects where requirements may evolve, and new risks may be identified as the project progresses. [The iterative nature of the Spiral Model ensures that the ATC system can be refined over time to meet the high safety and reliability standards required for air traffic control operations¹².](#)

List 2 difference between QA and QC?

Certainly! Here are two key differences between Quality Assurance (QA) and Quality Control (QC):

1. **Focus:**

- QA is a process-oriented approach that focuses on preventing defects by ensuring that the processes used in software development are adequate and followed correctly¹.
- QC is a product-oriented approach that focuses on identifying and fixing defects in the final product. It involves testing the product to ensure it meets the defined quality standards¹.

2. Activity Type:

- QA is a proactive measure that involves activities like process definition, process improvement, and training, which are aimed at improving the development process to prevent defects¹.
- QC is a reactive measure that involves activities like inspections, testing, and reviews, which are aimed at detecting and correcting defects in the already developed product¹.

These differences highlight that QA is about building the right process to create a quality product, while QC is about verifying that the product created meets the quality expectations.

Mark each statement as True or False? Justify.

- 1. Test cases may contain multiple valid conditions**
 - 2. Test cases may contain multiple invalid conditions**
 - 3. Test cases may contain both valid and invalid conditions**
 - 4. Test cases may contain more than 1 step.**
1. **True.** Test cases may contain multiple valid conditions to verify that the system behaves as expected under various scenarios that meet the requirements.
 2. **True.** Test cases may contain multiple invalid conditions to ensure that the system handles error conditions gracefully and does not fail unexpectedly.
 3. **True.** Test cases can contain both valid and invalid conditions, especially in boundary testing where the aim is to test the limits of acceptable inputs and verify the system's response to both sides of the boundary.
 4. **True.** Test cases often contain more than one step, particularly when the test scenario is complex and requires a sequence of actions to validate a particular functionality or feature.

Test cases are designed to cover a wide range of scenarios, including different combinations of input conditions and the steps required to execute them, to ensure comprehensive testing of the software system.

What are design patterns? What are the advantages of using design patterns? Explain any two design patterns.

Design patterns are general, reusable solutions to commonly occurring problems within a given context in software design¹. They are not finished designs that can be directly transformed into code but are templates for solving a problem that can be used in many different situations.

Advantages of using design patterns include:

- **Reusability:** Design patterns can be reused in multiple projects, which saves time and effort².
- **Speed Up Development Process:** They provide tested, proven development paradigms that speed up the development process³.
- **Reduce Common Problems:** Design patterns address common issues faced by developers, reducing the number of bugs and issues during development¹.
- **Improve Object-Oriented Skills:** They help developers to improve their object-oriented design skills¹.
- **Easy to Understand:** Design patterns make it easier for new developers to understand the existing code as they provide a common vocabulary¹.

Now, let's explain two design patterns:

1. Singleton Pattern:

- **Purpose:** Ensures that a class has only one instance and provides a global point of access to it⁴.
- **Use Case:** It's often used for managing connections to a database or a file system where having multiple instances could cause problems like inconsistent data or performance issues.
- **Example:**

```
public class DatabaseConnector {  
    private static DatabaseConnector instance;  
  
    private DatabaseConnector() {  
        // Private constructor to prevent instantiation.  
    }  
  
    public static DatabaseConnector getInstance() {  
        if (instance == null) {  
            instance = new DatabaseConnector();  
        }  
        return instance;  
    }  
}
```

2. Observer Pattern:

- **Purpose:** Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically⁵.
- **Use Case:** Useful in event handling systems, such as a user interface framework where changes in the user interface need to be reflected in other parts of the application.
- **Example:**
- ```

public interface Observer {
 void update();
}

public class ConcreteObserver implements Observer {
 public void update() {
 // Perform update actions.
 }
}

public class Subject {
 private List<Observer> observers = new ArrayList<>();

 public void attach(Observer observer) {
 observers.add(observer);
 }

 public void notifyObservers() {
 for (Observer observer : observers) {
 observer.update();
 }
 }
}

```

These patterns are part of a larger set of design patterns typically categorized into **Creational**, **Structural**, and **Behavioral** patterns, each serving a different aspect of software design and solving different types of problems<sup>6</sup>.

What is swimlanes? Using swimlanes draw activity diagram for examination software.

**Illustrate a method of constructing a class diagram. Draw a class diagram for the following case study.**

#### Case Study

A new school district operates a fleet of 40 buses that serves approximately 1000 students in grades KG to 12 std. The bus operation involves 30 regular routes plus special routes for activities, athletic events and summer session, the district employs 12 full time drivers and 25 to 30 part time drivers. A dispatcher co-ordinates the staffing and routes and relays messages to drivers regarding students and parents who call about pick up and drop-off arrangements.

Q.6b) Draw a state diagram for the online order delivery system. You must include the following:

- i) The states that must be included in the diagram are: checking, dispatching, waiting, cancelled and delivered (Additionally you may include more states)
- ii) Check whether all the items are available or not.
- iii) Include whether all the items are checked or not.
- iv) Include if some items are not in the stock.

Show appropriate transitions from a given state. Use the concept of Superstate

## **List 2 differences between software architecture and software design. If you were asked to follow Test Driven Development to develop Linux kernel, explain the steps to be followed**

Certainly! Here are two key differences between software architecture and software design:

1. **Scope:** Software architecture is about the fundamental structure of the entire system, focusing on the high-level organization of software components and the ways they interact<sup>1</sup>. In contrast, software design is concerned with the specification of individual software components, detailing their responsibilities, interfaces, and interactions within the larger system<sup>1</sup>.
2. **Focus:** Software architecture defines the core elements and properties required for the system's infrastructure, ensuring it meets technical and operational requirements<sup>1</sup>. Software design, however, delves into the implementation details of these components, including class definitions, functions, and their usages<sup>1</sup>.

As for Test Driven Development (TDD) in the context of Linux kernel development, the steps would be as follows:

1. **Add a Test:** Write a test that defines a function or improvements of a function, which should be very succinct.
2. **Run All Tests:** Run all tests and see if the new test fails. This validates that the test harness is working correctly, that the new test does not mistakenly pass without requiring any new code, and that the required feature does not already exist.

3. **Write the Code:** Write some code that causes the test to pass. The code written at this stage is not perfect and may require refinement.
4. **Run Tests:** Run the tests again. If they pass, the programmer can be confident that the code meets all the tested requirements.
5. **Refactor:** Refactor the code to remove duplication, improve the design, and accept further tests.
6. **Repeat:** Repeat the process for the next piece of functionality.

This TDD approach ensures that the Linux kernel is developed with a focus on passing tests, leading to better design, higher quality, and more maintainable code<sup>2</sup>.  
It's important to note that while TDD can be applied to Linux kernel development, it requires a deep understanding of the kernel's architecture and the ability to write tests that can effectively exercise the low-level components of the kernel<sup>3</sup>.

**Let us consider a satellite based mobile communication product.**

**Classify the risks into Project risk, Technical risk and Business risk.**

- a) **What if the project cost escalates and overshoots what was estimated?**
- b) **What if the mobile phones that are developed become too bulky in size to conveniently carry?**
- c) **What if it is later found out that level of radiation coming from the phone is harmful to human being?**
- d) **What if call hand-off between satellites becomes too difficult to implement?**
- e) **What if the key person leave in between the development?**

For a satellite-based mobile communication product, the risks can be classified as follows:

- a) **Project Risk:** If the project cost escalates and overshoots what was estimated, it poses a project risk. This could lead to budget overruns, project delays, and may require additional funding or scope reduction to accommodate the increased costs.
- b) **Technical Risk:** If the mobile phones that are developed become too bulky in size to conveniently carry, it represents a technical risk. This could impact the design and engineering aspects of the product, making it less competitive and appealing to consumers.
- c) **Business Risk:** If it is later found out that the level of radiation coming from the phone is harmful to human beings, it poses a significant business risk. This could lead to legal liabilities, loss of consumer trust, product recalls, and a negative impact on the company's reputation and profitability.
- d) **Technical Risk:** If call hand-off between satellites becomes too difficult to implement, it is a technical risk. This could affect the core functionality of the product, potentially leading to unreliable service and a failure to meet the product's intended performance criteria.
- e) **Project Risk:** If the key person leaves in between the development, it poses a project risk. The departure of a critical team member can lead to knowledge loss, disruption in the project timeline, and may necessitate the hiring and training of new personnel, which can be time-consuming and costly.

Each of these risks requires careful consideration and mitigation strategies to ensure the successful development and launch of the satellite-based mobile communication product.

## November 2020

**Attempt to suggest a process model for the 4 scenarios. Justify the chosen model**

- a) Requirements has a number of risk so has to include risk analysis as they develop the software
- b) Team size is less and but have excellent skill so team can prioritize

## **requirement and develop**

For the scenarios provided, here are the suggested process models and justifications:

- a) Requirements have a number of risks so has to include risk analysis as they develop the software:** The **Spiral Model** would be the most appropriate for this scenario. [It is a risk-driven process model that combines elements of both iterative and waterfall models with a focus on risk analysis<sup>1</sup>.](#) [The Spiral Model allows for iterative refinement through each spiral phase, with significant emphasis on identifying and mitigating risks at every step of development<sup>2</sup>.](#) This model is particularly useful when dealing with complex projects where risks need to be carefully managed and reduced.
- b) Team size is less but have excellent skill so team can prioritize requirement and develop:** For a small, highly skilled team, the **Agile Model** or specifically the **Scrum framework** within Agile would be suitable. [Agile methodologies, such as Scrum, are designed to accommodate changing requirements and prioritize tasks effectively<sup>3</sup>.](#) They allow small teams to work collaboratively, focus on delivering high-value features first, and adapt quickly to changes. [Scrum, in particular, empowers skilled teams to self-organize and make decisions about how to best approach work, which can lead to higher productivity and better project outcomes<sup>3</sup>.](#)

These models are chosen based on their ability to handle the specific challenges presented in each scenario. The Spiral Model's focus on risk management makes it ideal for projects with significant risks, while Agile and Scrum are well-suited for small, skilled teams due to their flexibility and emphasis on collaboration and prioritization.

---

For the given Event List (a) Draw the DFD level 1 diagram (Happy path)

(b) Sequence diagram for a single use case

- i. Customer books a cab from his/her smartphone via an app
  - ii. UBER dispatch algorithm sends a request to the driver nearest in proximity to customer
  - iii. Driver may accept or reject the ride request
  - iv. If driver decides to reject the request then system will attempt to connect the customer with next closest available driver
  - v. If driver accepts the request then system sends Driver Details and an estimation on how long your car will take to show up
  - vi. Driver picks up customer and drops at destination
  - vii. Application prepares Trip Bill and driver shows the same to
- 

customer

- viii. Customer verifies Trip Bill
- ix. Customer can pay in cash or electronically
- x. If customer makes payment electronically then payment made by customer is sent to accounts department. Account department keeps 20% of received amount and sends remaining to driver
- xi. Receipt is send to customer via email

---

Find the critical path. Tabulate EST, EFT, LST, LFT and FLOAT (CO3)

| Activity | Description        | Duration | Predecessors |
|----------|--------------------|----------|--------------|
| A        | Prepares Office    | 12       | -            |
| B        | Procure Equipments | 8        | -            |
| C        | Design tests       | 5        | -            |
| D        | Install Equipment  | 10       | A and B      |
| E        | Test Systems       | 10       | C and D      |
| F        | Train Users        | 5        | A and B      |

---

6. For the given code below find the cyclomatic complexity in all the three methods

```
int main()
{
 int n, index;
 cout << "Enter a number: " << endl;
 cin >> n;
 index = 2;
 while (index <= n - 1) {
 if (n % index == 0) {
 cout << "It is not a prime number" << endl;
 break;
 }
 index++;
 }
 if (index == n)
 cout << "It is a prime number" << endl;
} // end main
```

## **Choose the most appropriate architectural pattern**

**i. Wants a system that can be divided into reusable, loosely coupled**

**components that can be flexibly combined and arranged to transform**

**between various data formats.**

**ii. Wants a distributed system with a structure that enables that service**

**users do not need to know the nature or location of service providers.**

**iii. Wants a system that quickly can analyze enormous volumes of data**

**by sorting the data and then analyzing the grouped data.**

For the scenarios described, the most appropriate architectural patterns would be:

### i. Microservices Architecture:

- **Description:** This pattern structures an application as a collection of loosely coupled services, which implement business capabilities<sup>1</sup>. Each service is a small, independent process that communicates with other services through a well-defined interface using lightweight protocols<sup>1</sup>.
- **Justification:** Microservices architecture is ideal for systems that require reusable components. It allows for flexibility in combining and arranging services to transform between various data formats. Each service can be developed, deployed, and scaled independently<sup>1</sup>.

### ii. Service-Oriented Architecture (SOA):

- **Description:** SOA is an architectural pattern where services are provided to other components by application components, through a communication protocol over a network<sup>2</sup>. The key is that the service users do not need to know the nature or location of service providers<sup>2</sup>.
- **Justification:** SOA hides the details of service providers from service users, allowing for a distributed system where services can be consumed without knowing their exact implementation or location. This abstraction makes the system more flexible and scalable<sup>2</sup>.

### iii. MapReduce:

- **Description:** MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster<sup>3</sup>. The process involves two tasks - Map (sorting data) and Reduce (analyzing the sorted data)<sup>3</sup>.
- **Justification:** The MapReduce pattern is suitable for quickly analyzing enormous volumes of data. It allows for the data to be processed in parallel, significantly speeding up the analysis. This pattern is widely used in big data applications for its efficiency in handling large data sets<sup>3</sup>.

These architectural patterns provide strategies for addressing the specific needs of each scenario, ensuring that the system is designed in a way that supports its intended functionality and performance requirements.

## Give a scenario and explain any two design pattern

Design patterns are standard solutions to common problems in software design. They are templates that can be applied to real-world coding situations. Here are two common design patterns explained with scenarios:

**1. Singleton Pattern:** The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. This is particularly useful when exactly one object is needed to coordinate actions across the system.

**Scenario:** Imagine you're developing a printer spooler for an operating system. The spooler should manage all print jobs in a queue, ensuring that only one job is processed at a time. Implementing the printer spooler as a Singleton ensures that there is only one queue for all print jobs, preventing conflicts and ensuring proper management of printer resources.

```
class PrinterSpooler:
 _instance = None

 @staticmethod
 def getInstance():
 if PrinterSpooler._instance == None:
 PrinterSpooler()
 return PrinterSpooler._instance

 def __init__(self):
 if PrinterSpooler._instance != None:
 raise Exception("This class is a singleton!")
 else:
 PrinterSpooler._instance = self
 self.job_queue = []

 def add_job(self, job):
 self.job_queue.append(job)

 def process_jobs(self):
 while self.job_queue:
 job = self.job_queue.pop(0)
 # Process the job
 print(f"Printing: {job}")
```

**2. Observer Pattern:** The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Scenario:** Consider a weather station that tracks weather data and needs to update multiple display elements such as current conditions, weather statistics, and a forecast display whenever the weather data changes.

```
class WeatherStation:
 def __init__(self):
 self.observers = []
 self.temperature = 0
 self.humidity = 0
 self.pressure = 0

 def register_observer(self, observer):
 self.observers.append(observer)

 def remove_observer(self, observer):
 self.observers.remove(observer)
```

```

def notify_observers(self):
 for observer in self.observers:
 observer.update(self.temperature, self.humidity, self.pressure)

def measurements_changed(self):
 self.notify_observers()

def set_measurements(self, temperature, humidity, pressure):
 self.temperature = temperature
 self.humidity = humidity
 self.pressure = pressure
 self.measurements_changed()

class CurrentConditionsDisplay:
 def update(self, temperature, humidity, pressure):
 # Update the display with the latest measurements
 print(f"Current conditions: {temperature}F degrees and {humidity}% humidity")

Usage
weather_station = WeatherStation()
current_conditions_display = CurrentConditionsDisplay()
weather_station.register_observer(current_conditions_display)

Set new weather measurements
weather_station.set_measurements(70, 65, 30.4)

```

In the Singleton pattern, the key is controlling object creation, while in the Observer pattern, the focus is on the communication between objects<sup>12</sup>. These patterns are widely used because they solve common problems in a reliable and proven way.

## 5. An artist agreed to prepare portraits of 100 celebrities in 200 days at a

**price of \$500 per portrait. Fifty days later, artist finished 21 portraits with an**

**actual total cost of \$11,400. What is the status of the Project? Specify your**

**answer by making all possible EVA (Earned Value Analysis) calculations like**

**PV, AC, EV, SV, CV, SPI and CPI.Explain the factors that might delay a Project Schedule. State the product**

**metrics that will be affected.**

To assess the status of the project using Earned Value Analysis (EVA), we'll need to calculate the following metrics:

- **Planned Value (PV)**: The budgeted cost for the work scheduled to be completed by a certain date<sup>1</sup>.
- **Actual Cost (AC)**: The actual cost incurred for the work completed by the specified date<sup>1</sup>.
- **Earned Value (EV)**: The budgeted amount for the work actually completed by the specified date<sup>1</sup>.
- **Schedule Variance (SV)**: The difference between EV and PV<sup>1</sup>.
- **Cost Variance (CV)**: The difference between EV and AC<sup>1</sup>.
- **Schedule Performance Index (SPI)**: The ratio of EV to PV<sup>1</sup>.
- **Cost Performance Index (CPI)**: The ratio of EV to AC<sup>1</sup>.

Given:

- The total budget (BAC) for 100 portraits is ( $100 \times \$500 = \$50,000$ ).
- The project duration is 200 days.

After 50 days (which is 25% of the time), the artist should have completed 25% of the portraits, which is 25 portraits.

Calculations:

- **PV** = 25% of BAC = ( $0.25 \times \$50,000 = \$12,500$ )
- **AC** = \$11,400 (given)
- **EV** = 21 portraits completed ( $\times \$500/\text{portrait} = \$10,500$ )

Now, let's calculate the EVA metrics:

- **SV** = EV - PV =  $\$10,500 - \$12,500 = -\$2,000$
- **CV** = EV - AC =  $\$10,500 - \$11,400 = -\$900$
- **SPI** = EV / PV =  $\$10,500 / \$12,500 = 0.84$
- **CPI** = EV / AC =  $\$10,500 / \$11,400 \approx 0.92$

The negative SV indicates the project is behind schedule, and the negative CV indicates the project is over budget. The SPI less than 1 suggests the project is progressing at 84% of the rate originally planned, and a CPI less than 1 indicates that for every dollar spent, only \$0.92 worth of work is being completed.

Factors that might delay a Project Schedule include:

- **Poor Planning**: Inadequate planning can lead to unrealistic timelines and resource allocation<sup>2</sup>.
- **Scope Creep**: Uncontrolled changes or continuous growth in a project's scope<sup>2</sup>.
- **Resource Constraints**: Limitations on personnel, equipment, or materials<sup>2</sup>.

- **Poor Communication:** Inadequate communication among team members<sup>2</sup>.
- **Risk Management Issues:** Unidentified or unplanned risks<sup>2</sup>.
- **Task Dependencies:** Delays in one task can affect dependent tasks<sup>2</sup>.
- **Task Complexity:** Underestimation of the complexity of tasks<sup>2</sup>.
- **Lack of Skills or Training:** Team members lacking necessary skills or training<sup>2</sup>.

Product metrics that will be affected by a project delay include:

- **Time to Market:** Delays can push back the release date of the product.
- **Budget:** Overruns can occur due to extended timelines.
- **Quality:** Rushing to meet deadlines can lead to compromised quality.
- **Customer Satisfaction:** Delays can lead to dissatisfaction and loss of trust.
- **Resource Utilization:** Inefficient use of resources due to extended project duration<sup>3</sup>.

These metrics are crucial for assessing the impact of delays on the overall success and viability of a product.

## **What is the best way of doing software integration Justify**

### **your answer**

The best way of doing software integration involves adhering to a set of best practices that ensure the process is efficient, reliable, and aligns with the business strategy. Here are some key practices to follow:

1. **Start with a Clear Understanding of the Business Strategy and Roadmap:** Before integrating software systems, it's crucial to have a clear understanding of the business objectives and how the integration will support them<sup>1</sup>.
2. **Decide on the Integration Pattern and Approach:** Choose an integration pattern that fits the needs of the project, whether it's point-to-point, hub-and-spoke, or a more complex service-oriented architecture<sup>1</sup>.
3. **Collaborate with Relevant Stakeholders:** Engage all relevant stakeholders, including business analysts, developers, and end-users, to ensure that the integration meets the needs of all parties<sup>1</sup>.
4. **Whiteboard/Brainstorm All Possible Business Scenarios:** Consider all possible use cases and scenarios to ensure that the integration covers all necessary functionalities and handles edge cases<sup>1</sup>.
5. **Choose the Right Implementation Team:** The team should have the right mix of skills and experience to handle the technical and business aspects of the integration<sup>1</sup>.

Justification for these practices includes:

- **Alignment with Business Goals:** Ensuring that the integration supports the overall business strategy and adds value.
- **Technical Suitability:** Selecting the right integration pattern ensures that the system is scalable, maintainable, and performs well.
- **Stakeholder Satisfaction:** Involving stakeholders helps in creating a system that truly meets user needs and expectations.
- **Comprehensive Coverage:** Brainstorming scenarios ensures that the system is robust and can handle real-world use cases.
- **Successful Implementation:** A skilled team is essential for overcoming technical challenges and delivering a successful integration.

These best practices are critical for successful software integration, as they help in creating a system that is well-designed, functional, and aligned with business objectives. They are preferable over ad-hoc or unstructured approaches, which can lead to integration issues, technical debt, and systems that do not meet user needs or business goals.