

1. Design agile process of your own department(web application/iot application/ robotics application/construction)

A: Agile Process for Web Application Development Department

1. Team Structure:

- **Roles:** Product Owner (PO), Scrum Master (SM), Development Team (developers, designers, QA), Stakeholders.
- **Team Size:** 5-9 members.

2. Agile Ceremonies:

- **Sprint Planning** (bi-weekly): Define sprint goals, select and break down backlog items.
- **Daily Stand-up** (daily): Share progress, identify blockers, plan the day.
- **Sprint Review** (end of each sprint): Demonstrate work, gather feedback.
- **Sprint Retrospective** (end of each sprint): Reflect on the sprint, discuss improvements.
- **Backlog Refinement** (ongoing): Ensure backlog items are clear, prioritized, and estimated.

3. Workflow:

- **Backlog Grooming:** Regularly update and prioritize the product backlog.
- **Development:** Commit code frequently, conduct peer reviews and automated testing.
- **Testing:** Use continuous integration and manual testing for critical features.
- **Sprint Review and Retrospective:** Demonstrate work and reflect on the process for continuous improvement.

4. Tools and Technologies:

- **Project Management:** JIRA, Trello.
- **Version Control:** Git, GitHub/GitLab.
- **Communication:** Slack, Zoom.
- **Testing:** Jenkins, Selenium.
- **Design:** Figma.
- **Documentation:** Confluence, Markdown.

This process ensures iterative development, continuous feedback, and collaboration, essential for the dynamic nature of web application projects.

2. Illustrate with an example the task of collecting requirements at various steps

A: **Project Example:** Developing a new e-commerce web application.

1. Initial Requirement Gathering:

- **Activity:** Stakeholder Interviews.
- **Example:** The Product Owner (PO) meets with the client to discuss key features like user registration, product catalog, and checkout process. High-level requirements are gathered, such as “Users need to create accounts” and “Customers should be able to search for products.”

2. Creating and Refining the Product Backlog:

- **Activity:** Backlog Grooming.
- **Example:** The PO converts the initial requirements into user stories like “As a user, I want to register so I can make purchases.” The development team then refines these stories, breaking them into smaller tasks and estimating their complexity.

3. Sprint Planning:

- **Activity:** Defining Sprint Goals.
- **Example:** During sprint planning, the team selects the user story “User registration” and identifies specific tasks like “Design registration form” and “Implement email verification.” They discuss acceptance criteria and clarify any ambiguities with the PO.

4. Ongoing Refinement and Feedback:

- **Activity:** Sprint Reviews and Continuous Feedback.
- **Example:** At the end of the sprint, the team presents the registration feature to the client. The client requests an additional “password strength indicator.” The requirement is added to the backlog and planned for a future sprint.

3. how is debugging different from testing

A: **Testing** is the process of identifying defects in software by comparing the actual behavior of the software against the expected behavior. It involves running test cases to find errors or issues and is usually performed by QA testers. For example, a tester might run automated tests to check if a feature behaves as specified.

Debugging is the process of locating, diagnosing, and fixing these identified defects. It involves analyzing the code to understand why an error occurred and then correcting it. Debugging is typically done by developers using tools like debuggers to step through code and identify the cause of issues.

While testing aims to discover problems, debugging is focused on resolving them. Testing provides a list of issues to be addressed, whereas debugging is about modifying the code to fix these issues, ensuring the software functions correctly.

4. The software is integral part of human life. substantiate the statement using evolving role of software.

A: The Integral Role of Software in Human Life

Software has become a fundamental part of daily life, deeply embedded in numerous aspects of modern society. Initially, software was used for basic computational tasks and scientific research. Today, its role has evolved to influence nearly every facet of human life.

1. **Communication and Connectivity:** Software underpins communication technologies, facilitating instant messaging, video calls, and social media interactions across the globe. Applications like WhatsApp, Zoom, and Facebook have transformed how people connect, breaking geographical barriers and enabling real-time communication.
 2. **Healthcare:** In healthcare, software is crucial for managing patient records, diagnosing diseases, and even performing surgeries. Medical software like electronic health records (EHR) systems improves patient care by ensuring accurate and timely information sharing, while telemedicine platforms provide remote healthcare services, enhancing accessibility.
 3. **Education:** The education sector relies heavily on software for virtual learning environments, educational apps, and online courses. Platforms like Khan Academy and Coursera offer learners around the world access to high-quality education and resources, fostering lifelong learning and knowledge sharing.
 4. **Daily Convenience:** Everyday activities such as shopping, banking, and entertainment have been revolutionized by software. E-commerce platforms like Amazon, online banking services, and streaming services like Netflix provide convenience and efficiency, simplifying tasks that once required significant time and effort.
-

5. Write the importance of agile methodology and Explain that how is the scrum meeting effective technique than traditional software model.

A: Importance of Agile Methodology and the Effectiveness of Scrum Meetings

Agile methodology is crucial in modern software development due to its ability to adapt to changing requirements, foster collaboration, and deliver value incrementally. Unlike traditional software models that rely on extensive upfront planning and documentation, Agile promotes iterative development, allowing teams to respond quickly to feedback and deliver functional software in shorter timeframes.

Scrum meetings, a key component of Agile, are particularly effective in contrast to traditional software models for several reasons:

1. **Regular Communication and Collaboration:** Scrum meetings, including the daily stand-up, sprint planning, review, and retrospective, facilitate regular communication among team members. Unlike traditional models where communication might be limited to formal meetings, Scrum encourages daily interactions, enabling quick problem-solving and alignment on project goals.
2. **Adaptability to Change:** Agile methodologies, including Scrum, embrace change as a natural part of the development process. By breaking work into smaller, manageable iterations (sprints),

Scrum allows teams to adapt to changing requirements and priorities more effectively than traditional models. This flexibility enables teams to deliver valuable features incrementally, ensuring that the product meets evolving customer needs.

3. **Focus on Delivering Value:** Scrum meetings emphasize the delivery of working software at the end of each sprint. This iterative approach ensures that the team focuses on delivering tangible value to the customer with each iteration, rather than getting bogged down in extensive planning and documentation. In contrast, traditional models often prioritize comprehensive upfront planning, which may result in delayed delivery and an increased risk of delivering features that do not align with customer expectations.
 4. **Continuous Improvement:** The sprint retrospective, a key Scrum meeting, promotes a culture of continuous improvement within the team. By reflecting on what went well, what didn't, and how to improve, teams can identify and address issues proactively, leading to higher productivity and quality over time. This focus on continuous improvement is a core principle of Agile methodologies and sets them apart from traditional software development models, which may lack mechanisms for feedback and reflection.
-

6. Write a brief note on component based development process model.

A: Component-Based Development Process Model

Component-based development (CBD) is a software engineering approach where systems are built by composing pre-existing software components. In this model, software functionality is encapsulated into reusable components, which are then integrated to create larger systems or applications. The CBD process typically involves the following key steps:

1. **Component Identification:** The first step in CBD is identifying reusable software components that encapsulate specific functionality or services. These components can be sourced from existing libraries, third-party vendors, or developed in-house.
 2. **Component Selection and Integration:** Once identified, suitable components are selected based on their compatibility with the project requirements. Integration involves combining these components to form the desired system architecture. This may involve adapting or customizing components to fit the project's needs.
 3. **Testing and Verification:** After integration, thorough testing and verification are conducted to ensure that the components work together as expected and meet the system requirements. This includes testing individual components for functionality, as well as testing the integrated system for performance, reliability, and security.
 4. **Deployment and Maintenance:** Once tested and verified, the system is deployed for use by end-users. Maintenance involves ongoing support, updates, and enhancements to the system over its lifecycle. CBD facilitates easier maintenance and scalability since changes can be made at the component level without affecting the entire system.
-

7. Draw a sample flow graph and determine the cyclomatic complexity.

A: Start

|

V

A ----> B ----> C



In this flow graph:

- Nodes represent individual statements or blocks of code.
- Arrows represent control flow between nodes.
- Decision points (if statements) create branches in the flow.

To calculate the cyclomatic complexity ($V(G)$) of this flow graph, you can use the formula:

mathematica

Copy code

$$V(G) = E - N + 2P$$

Where:

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

For our example:

- $E = 7$ (number of arrows)
- $N = 6$ (number of nodes)
- $P = 1$ (single program)

Now, plug the values into the formula:

scss

Copy code

$$\begin{aligned}
 V(G) &= 7 - 6 + 2(1) \\
 &= 7 - 6 + 2 \\
 &= 3
 \end{aligned}$$

So, the cyclomatic complexity of this flow graph is 3.

8. A program state the following for an input field: The program shall accept an input value [4] of 4 digit integer equal or greater than 2000 and less than or equal 8000. Determine the test cases using

- i) Equivalence class partitioning
- ii) Boundary value analysis

A: Test Cases using Equivalence Class Partitioning and Boundary Value Analysis

Equivalence Class Partitioning:

Equivalence class partitioning divides the input domain into classes or groups of data that are expected to be processed in the same way by the system. For the given input field, we can identify the following equivalence classes:

1. **Valid Input:**
 - Equivalence class for a 4-digit integer between 2000 and 8000, inclusive.
 - Examples: 2000, 4500, 8000.
2. **Invalid Input:**
 - Equivalence class for inputs outside the valid range (less than 2000 or greater than 8000).
 - Examples: 1999, 8001, -1000.

Boundary Value Analysis:

Boundary value analysis focuses on testing around the boundaries of equivalence classes. For a 4-digit integer between 2000 and 8000, the boundaries are:

1. **Lower Boundary:**
 - Smallest valid input value: 2000.
 - One less than the smallest valid input value: 1999.
2. **Upper Boundary:**
 - Largest valid input value: 8000.
 - One greater than the largest valid input value: 8001.

Test Cases:

Using Equivalence Class Partitioning:

1. **Valid Input Test Cases:**
 - Test case 1: Input = 4500 (within the valid range).
 - Test case 2: Input = 2000 (lower boundary).
 - Test case 3: Input = 8000 (upper boundary).
2. **Invalid Input Test Cases:**
 - Test case 4: Input = 1999 (below the lower boundary).
 - Test case 5: Input = 8001 (above the upper boundary).
 - Test case 6: Input = -1000 (outside the valid range).

Using Boundary Value Analysis:

1. **Boundary Test Cases:**

- Test case 7: Input = 1999 (just below the lower boundary).
 - Test case 8: Input = 2000 (lower boundary).
 - Test case 9: Input = 8000 (upper boundary).
 - Test case 10: Input = 8001 (just above the upper boundary).
-

9. Write the metrics used for software maintenance.

A: Metrics for Software Maintenance

1. **Mean Time to Repair (MTTR):**

- Formula: $MTTR = \text{Total downtime} / \text{Number of repairs}$
- Example: If a software system experiences a total downtime of 10 hours due to 5 repairs, then the MTTR would be $10 \text{ hours} / 5 \text{ repairs} = 2 \text{ hours per repair}$.

2. **Mean Time Between Failures (MTBF):**

- Formula: $MTBF = \text{Total uptime} / \text{Number of failures}$
- Example: If a software system operates continuously for 100 hours before experiencing 2 failures, then the MTBF would be $100 \text{ hours} / 2 \text{ failures} = 50 \text{ hours per failure}$.

3. **Percentage of Defects Detected and Fixed (Defect Removal Efficiency):**

- Formula: $\text{Defect Removal Efficiency} = (\text{Number of defects detected and fixed during maintenance} / \text{Total number of defects}) * 100\%$
- Example: If during maintenance, 80 defects out of 100 total defects are detected and fixed, then the Defect Removal Efficiency would be $(80 / 100) * 100\% = 80\%$.

4. **Percentage of Change Requests Implemented (Change Implementation Success Rate):**

- Formula: $\text{Change Implementation Success Rate} = (\text{Number of successfully implemented change requests} / \text{Total number of change requests}) * 100\%$
 - Example: If out of 50 change requests submitted, 40 are successfully implemented, then the Change Implementation Success Rate would be $(40 / 50) * 100\% = 80\%$.
-

10. What is defect removal efficiency? How is it used to measure software quality?

A: **Defect removal efficiency (DRE)** measures the effectiveness of the software development process in identifying and fixing defects. It is calculated as the ratio of defects detected and fixed during a specific phase of development or maintenance to the total number of defects identified across all phases.

Formula: $DRE = (\text{Number of defects detected and fixed}) / (\text{Total number of defects}) * 100\%$


For example, if during testing, 80 defects out of 100 total defects are identified and fixed, the DRE would be calculated as follows:


$$DRE = (80 / 100) * 100\% = 80\%$$


Defect removal efficiency provides insights into the quality of the software development process. A higher DRE indicates that a larger proportion of defects are being detected and fixed before the


software is released, leading to higher software quality and reliability. Conversely, a lower DRE suggests that many defects are escaping detection and may lead to quality issues in the delivered software. Therefore, DRE is a critical metric used to assess and improve the effectiveness of the software development process and ultimately enhance software quality.

11. Differentiate between incremental model and iterative model

A:  Incremental model focuses on delivering functionality in small, incremental chunks, while iterative model focuses on refining the entire software system through repeated cycles of development.

 Incremental model delivers partial functionality in each increment, whereas iterative model delivers a complete version of the software at the end of each iteration.

 Incremental model emphasizes delivering value incrementally and receiving feedback after each increment, while iterative model emphasizes continuous refinement based on ongoing feedback throughout each iteration.

 In incremental model, each increment builds upon the previous one, while in iterative model, each iteration produces a new version of the software with added features or improvements.

12. Describe SCRUM model

A: **Scrum** is an Agile framework used for managing software development projects. It emphasizes iterative development, collaboration, and flexibility to adapt to changing requirements. Here's a concise overview of the Scrum model:

1. Roles:

- **Product Owner:** Represents stakeholders and prioritizes the product backlog.
- **Scrum Master:** Facilitates the Scrum process and removes impediments.
- **Development Team:** Self-organizing cross-functional team responsible for delivering the product increment.

2. Artifacts:

- **Product Backlog:** List of all desired features, enhancements, and bug fixes prioritized by the Product Owner.
- **Sprint Backlog:** Subset of items from the Product Backlog selected for implementation in the current sprint.
- **Increment:** Potentially shippable product increment created during the sprint.

3. Events:

- **Sprint Planning:** Collaborative meeting at the start of each sprint to select backlog items and plan the work.
- **Daily Stand-up:** Daily meeting for the Development Team to synchronize activities and identify any impediments.
- **Sprint Review:** Review of the completed increment with stakeholders at the end of the sprint to gather feedback.
- **Sprint Retrospective:** Reflection meeting at the end of the sprint to identify improvements for the next sprint.

4. Iterations:

- Scrum operates in fixed-length iterations called sprints, typically lasting 2-4 weeks.

- Each sprint aims to deliver a potentially shippable product increment, with feedback gathered at the end to inform future iterations.
 - The process is iterative, with the product backlog refined and reprioritized based on feedback and changing requirements.
-

13. Analyze various metrics for maintenance

A: ? **Mean Time to Repair (MTTR):**

- Formula: $MTTR = \text{Total downtime} / \text{Number of repairs}$

? **Mean Time Between Failures (MTBF):**

- Formula: $MTBF = \text{Total uptime} / \text{Number of failures}$

? **Defect Removal Efficiency (DRE):**

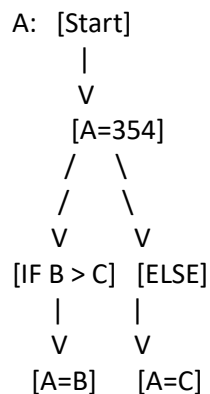
- Formula: $DRE = (\text{Number of defects detected and fixed}) / (\text{Total number of defects}) * 100\%$

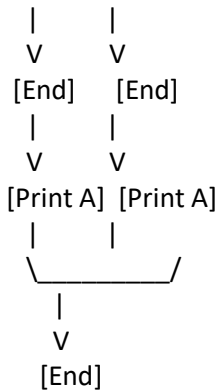
? **Percentage of Change Requests Implemented (Change Implementation Success Rate):**

- Formula: $\text{Change Implementation Success Rate} = (\text{Number of successfully implemented change requests}) / (\text{Total number of change requests}) * 100\%$
-

14. Design flow graph and find Cyclomatic complexity for the given code-

```
IF A = 354
THEN IF B > C
THEN A = B
ELSE A = C
END IF
END IF
PRINT A
```





In this flow graph:

- There are 7 nodes: Start, A=354, IF B > C, A=B, A=C, Print A, End.
- There are 8 edges connecting these nodes.
- There is 1 connected component.

Using the formula $V(G) = E - N + 2P$ $V(G) = 8 - 7 + 2(1) = 3$

So, the cyclomatic complexity $V(G)$ for the given code is 3.

15. describe the difference between process and project metrics

A: **Focus:** Process metrics focus on the development process itself, while project metrics focus on the performance and progress of individual projects or tasks within the process.

Purpose: Process metrics measure the efficiency and effectiveness of processes, while project metrics assess the performance and progress of specific projects or tasks.

Examples: Process metrics include indicators related to productivity, quality, and adherence to standards, while project metrics include indicators related to project schedule, budget, scope, and resource utilization.

Benefit: Process metrics help organizations improve overall development efficiency and quality by identifying areas for improvement in processes, while project metrics help project managers track project progress and make necessary adjustments to ensure successful project delivery within constraints.

16. how to overcome the disadvantages of waterfall model using agile process model

A: **Emphasizing iterative development.**

Enhancing customer collaboration.

Forming cross-functional teams.

Promoting continuous improvement.

17. Waterfall model is not suitable for real time system development. justify with a example

A: ☒ The Waterfall model's sequential nature doesn't accommodate changing requirements well.

☒ In real-time systems like power grid monitoring, requirements can evolve rapidly.

☒ Waterfall's fixed phases make it difficult to incorporate new requirements mid-development.

☒ Agile methodologies like Scrum offer iterative development and adaptability, better suited for real-time system development.

18. Develop the Extreme programming (XP) concept for the case study. E-vote website details are given in the case study. The categories of users are: Registered users: The users who have registered and can cast vote. Authorized user: Each user can cast vote exactly once. The major activities in the website are, Login/Sign up. Casting a vote.

A: **Extreme Programming (XP) Concept for E-Vote Website**

1. User Stories and Planning Game:

- Define user stories for login/sign up and casting a vote functionalities.
- Prioritize user stories based on customer feedback and business value.
- Conduct planning game to estimate and commit to user stories for each iteration.

2. Pair Programming:

- Implement login/sign up functionality using pair programming.
- One developer writes code while the other reviews, provides feedback, and suggests improvements.
- Rotate pairs frequently to share knowledge and maintain code quality.

3. Test-Driven Development (TDD):

- Write automated unit tests for login/sign up and casting a vote functionalities before writing code.
- Ensure tests fail initially, then implement code to pass tests.
- Refactor code continuously to improve design and maintainability.

4. Continuous Integration (CI):

- Set up CI server to automatically build and test code changes after every commit.
- Run automated regression tests to ensure new changes don't break existing functionality.
- Fix any issues promptly to keep the codebase stable and deployable.

5. Small Releases:

- Deploy small, incremental updates to the e-vote website frequently.
- Allow registered users to login/sign up and cast votes in the initial release.
- Iterate based on user feedback to enhance features and address any issues.

6. Refactoring:

- Continuously refactor code to improve clarity, maintainability, and performance.
- Eliminate duplication, simplify complex code, and adhere to coding standards.
- Use code reviews and pair programming to ensure refactored code meets quality standards.

7. Collective Code Ownership:

- Encourage all team members to take ownership of the codebase.


- Foster collaboration and knowledge sharing to facilitate smooth code maintenance and updates.
- Ensure documentation and comments are clear and up-to-date for shared understanding.

8. On-Site Customer:

- Collaborate closely with stakeholders, including registered and authorized users, throughout the development process.
 - Gather feedback during each iteration to validate assumptions and prioritize features.
 - Use feedback to make informed decisions and deliver value aligned with user needs.
-

19. differentiate between white box testing and black box testing

A:

Criteria	White Box Testing	Black Box Testing
Knowledge Required	Requires knowledge of internal code structure.	Requires no knowledge of internal code structure.
Focus	Tests internal logic, paths, and code branches.	Tests functionality, inputs, and outputs.
Design Techniques	Techniques include statement coverage, branch coverage, and path coverage.	Techniques include equivalence partitioning, boundary value analysis, and state transition testing.
Test Cases	Designed based on code structure and logic.	Designed based on requirements and specifications.
Testers	Developers or testers with programming skills.	Testers without programming skills can also perform testing.
Debugging	Easier to pinpoint defects and debug code.	Defects are identified based on observed behavior and may be more challenging to locate in the code.
Types of Testing	Includes unit testing, integration testing, and code review. 	Includes functional testing, system testing, and acceptance testing.

20. Design flow graph to find out paths, Regions and calculate Cyclomatic complexity for the given code-

```

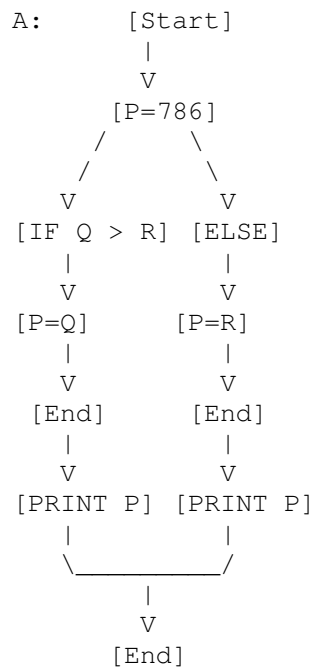
IF P = 786
THEN IF Q > R
THEN P = Q
ELSE P = R
END IF

```

```

END IF
PRINT P

```



In this flow graph:

- There are 7 nodes: Start, P=786, IF Q > R, P=Q, P=R, Print P, End.
- There are 8 edges connecting these nodes.
- There is 1 connected component.

Using the formula $V(G) = E - N + 2P$ $V(G) = 8 - 7 + 2(1) = 3$

So, the cyclomatic complexity $V(G)$ for the given code is 3.

21. What are the future trends in software metrics and how they can be used to support software development practices such as Agile and DevOps?

A: Future trends in software metrics include:

1. **Predictive Analytics:** Utilizing advanced analytics and machine learning techniques to predict software quality, identify potential risks, and optimize development processes.
2. **Real-time Monitoring:** Implementing tools and techniques for real-time monitoring of software development processes, allowing teams to identify issues promptly and make data-driven decisions.
3. **Continuous Feedback Loop:** Establishing a continuous feedback loop by integrating metrics into Agile and DevOps practices, enabling teams to continuously evaluate performance, iterate, and improve.

4. **Automation and AI:** Leveraging automation and artificial intelligence to collect, analyze, and interpret software metrics, reducing manual effort and providing actionable insights to support decision-making.

These future trends in software metrics can support Agile and DevOps practices by:

- **Enabling data-driven decision-making:** By providing actionable insights into software development processes, metrics empower teams to make informed decisions and prioritize activities effectively.
- **Facilitating continuous improvement:** Metrics help teams identify areas for improvement and track progress over time, enabling iterative refinement of Agile and DevOps practices.
- **Enhancing collaboration and transparency:** By promoting visibility into project status and performance metrics, teams can collaborate more effectively and align efforts towards common goals.
- **Supporting DevOps automation:** Integrating metrics into DevOps pipelines enables automated monitoring and analysis of software quality and performance, facilitating rapid feedback and continuous delivery.

Overall, the future trends in software metrics hold significant potential to enhance Agile and DevOps practices by promoting data-driven decision-making, continuous improvement, collaboration, and automation.
