1. List down all models of SDLC.

Answer:

<https://medium.com/@vghadigaokar/models-of-stlc-c8830011e7dd>

Here are some commonly known models of the Software Development Life Cycle (SDLC):

1. Waterfall Model: The Waterfall model follows a linear and sequential approach, where each phase is completed before moving to the next. It includes requirements gathering, design, development, testing, deployment, and maintenance.

2. Agile Model: Agile methodologies, such as Scrum or Kanban, focus on iterative and incremental development. They emphasize collaboration, adaptability to change, and regular feedback from stakeholders.

3. Spiral Model: The Spiral model combines elements of the Waterfall and iterative approaches. It involves multiple iterations, each consisting of planning, risk analysis, development, and evaluation. It incorporates feedback and risk assessment throughout the development process.

4. V-Model: The V-Model is an extension of the Waterfall model. It emphasizes the relationship between each development phase and its corresponding testing phase. Testing activities are planned and executed in parallel with each development stage.

5. Iterative Model: The Iterative model involves repeating cycles of development and testing in small increments. Each iteration includes requirements gathering, design, development, and testing. Feedback and learning from each iteration inform subsequent cycles.

6. Spiral Model: The Spiral model combines iterative and risk-driven approaches. It includes iterative cycles of development, along with risk analysis and mitigation. It focuses on managing risks and incorporating learning from previous cycles.

7. Prototype Model: The Prototype model involves the creation of a working prototype or mock-up of the software. Feedback from stakeholders is used to refine and improve the prototype, which then serves as a basis for development.

8. RAD (Rapid Application Development) Model: RAD is an iterative and accelerated model that emphasizes rapid prototyping and early user feedback. It involves a series of workshops or collaborative sessions to quickly develop and refine the software.

9. Incremental Model: The Incremental model involves the delivery of functional increments or modules of the software in multiple stages. Each increment builds upon the previous one, adding new features or functionalities.

10. DevOps Model: DevOps focuses on the integration of development and operations processes, emphasizing collaboration, automation, continuous integration, and continuous delivery. It aims to streamline software development and deployment.

It's important to note that different organizations and projects may adopt or adapt these models based on their specific needs, requirements, and development methodologies. Additionally, hybrid models can be created by combining elements from different models to suit unique project requirements.

1. What is STLC ? Also explain all stages of STLC.

Answer:

<https://medium.com/@vghadigaokar/what-is-stlc-8e19cc51fe80>

STLC stands for Software Testing Life Cycle. It is a series of activities and processes performed to ensure that the software being developed meets the specified requirements, is of high quality, and functions as intended. The STLC encompasses the planning, preparation, execution, and reporting of software testing activities throughout the software development life cycle (SDLC). Here are the stages of STLC:

1. **Requirement Analysis**: In this stage, testers analyze the requirements documentation to understand the software's functional and non-functional aspects. They identify any ambiguities, inconsistencies, or gaps in the requirements that may impact testing.

2. **Test Planning**: Test planning involves defining the overall testing strategy, objectives, scope, and timelines. Testers determine the test levels, identify test deliverables, and allocate resources. They create a comprehensive test plan that outlines the approach, test objectives, test environments, test data, and test schedules.

3. **Test Case Development**: Test cases are derived from the requirements and design documents. Testers develop detailed test cases that cover various scenarios, including positive and negative test cases. Test data, expected results, and preconditions are specified for each test case.

4**. Test Environment Setup**: Testers set up the required test environments, including hardware, software, and network configurations. They ensure that the test environment replicates the production environment as closely as possible to ensure accurate testing.

5. **Test Execution**: Test execution involves running the test cases as per the test plan. Testers execute the test cases, record the actual results, and compare them against the expected results. They report and track any defects encountered during the testing process.

6. **Defect Tracking**: Testers log defects for any deviations from the expected results. Defects are assigned priorities, severity levels, and are tracked using a defect tracking tool. Testers work closely with developers to resolve the reported defects.

7. **Test Reporting**: Testers generate various test reports that provide insights into the testing progress, defect status, and overall quality of the software. Test summary reports, defect reports, and test coverage reports are created to communicate the testing results to stakeholders.

8**. Test Closure**: In this final stage, the testing activities are reviewed, and the test closure report is prepared. Test closure involves evaluating the test coverage, identifying areas of improvement, and documenting lessons learned. Test artifacts, such as test cases and test data, are archived for future reference.

It's important to note that the STLC stages may vary slightly depending on the organization, project, and testing approach adopted. However, the core objectives of planning, designing, executing, and reporting the testing activities remain consistent across different STLC models.

1. As a test lead for web based application, your manager has asked you to identify and explain the different risk factors that should be included in the test plan. Can you provide a list of potential risks and their explanations that you would include in the test plan?

Answer:

<https://medium.com/@vghadigaokar/the-different-risk-factors-that-should-be-included-in-the-test-plan-a1f7176296c0>

When creating a test plan for a web-based application, it's important to consider various risk factors that could potentially impact the quality and success of the testing effort. Here is a list of potential risk factors and their explanations that you can include in the test plan:

**1. Compatibility Risks**: The risk of the application not functioning properly across different browsers, operating systems, or devices. This can lead to inconsistencies in the user experience and functionality.

**2. Security Risks**: The risk of vulnerabilities or weaknesses in the application's security controls, potentially leading to unauthorized access, data breaches, or privacy violations.

**3. Performance Risks**: The risk of the application not meeting performance requirements, such as slow response times, high resource consumption, or scalability issues. This can impact user satisfaction and overall system stability.

**4. Usability Risks:** The risk of the application being difficult to use or understand by its intended users. This can result in lower user adoption, increased support calls, and decreased user satisfaction.

**5. Functional Risks:** The risk of the application not meeting the specified functional requirements. This can lead to incorrect calculations, missing or incomplete features, and incorrect system behavior.

**6. Data Integrity Risks:** The risk of data corruption, loss, or inaccurate data processing within the application. This can impact the reliability and trustworthiness of the application's data.

**7. Integration Risks:** The risk of issues arising when integrating the application with other systems or third-party components. This includes data exchange problems, compatibility issues, or communication failures.

**8. Recovery and Backup Risks:** The risk of inadequate backup procedures or unreliable recovery mechanisms. This can result in data loss, extended downtime, or failure to restore the application to its normal state after a failure.

**9. Change Management Risks:** The risk of introducing defects or regressions when making changes or updates to the application. This includes issues related to version control, code merges, and release management.

**10. Resource Risks:** The risk of insufficient resources, such as time, budget, or skilled personnel, which may impact the testing effort's effectiveness and efficiency.

**11. Legal and Compliance Risks:** The risk of non-compliance with legal, regulatory, or industry-specific requirements, such as data protection regulations or accessibility standards.

**12. Environmental Risks:** The risk of issues related to the infrastructure, network, or hosting environment that could affect the availability and performance of the application.

1. Your TL( Team Lead) has asked you to explain the difference between quality assurance (QA) and quality control (QC) responsibilities. While QC activities aim to identify defects in actual products, your TL is interested in processes that prevent defects. How would you explain the distinction between QA and QC responsibilities to your boss?

Answer:

<https://medium.com/@vghadigaokar/quality-assurance-quality-control-is-it-same-b2c20a37fc6b>

Here's how I would explain the distinction between Quality Assurance (QA) and Quality Control (QC) responsibilities to my boss:

**Quality Assurance (QA) Responsibilities:**

1. QA focuses on preventing defects and ensuring that the development process is effective in producing a high-quality product.

2. QA activities are proactive and aim to establish processes, standards, and methodologies that contribute to the overall quality of the product.

3. QA involves activities such as defining and implementing quality standards, developing test strategies, and creating quality plans.

4. QA is involved in the early stages of the software development life cycle (SDLC) to identify potential risks, set quality goals, and establish guidelines for the development team.

5. QA responsibilities include conducting reviews and audits to ensure compliance with established processes, standards, and best practices.

6. QA team collaborates with stakeholders, including developers, business analysts, and project managers, to identify quality requirements, establish metrics, and monitor process improvements.

7. The ultimate goal of QA is to build a culture of quality, prevent defects from occurring, and continuously improve the development process.

**Quality Control (QC) Responsibilities:**

1. QC focuses on identifying defects and ensuring that the product meets the specified quality requirements.

2. QC activities are reactive and involve the testing and inspection of the actual product to detect any deviations from the expected quality standards.

3. QC involves activities such as executing test cases, conducting inspections, and performing various testing techniques to identify defects and verify the product's functionality.

4. QC is typically carried out during the later stages of the SDLC, such as system testing, regression testing, and user acceptance testing.

5. QC responsibilities include tracking and reporting defects, analyzing their root causes, and collaborating with development teams to resolve them.

6. QC team is responsible for ensuring that the product meets the defined quality criteria and complies with customer requirements and industry standards.

7. The goal of QC is to validate the product's quality, verify that it functions as intended, and deliver a defect-free product to the end-users.

In summary, QA focuses on preventing defects through establishing processes and standards, while QC focuses on identifying defects through testing and inspection of the product. QA activities aim to improve the overall development process, while QC activities focus on ensuring the quality of the final product.

1. Difference between manual testing and automation testing.

Answer:

<https://medium.com/@vghadigaokar/1-difference-between-manual-testing-and-automation-testing-16fdd8581b94>

Manual testing and automation testing are two different approaches to software testing, each with its own advantages and use cases. Here's a comparison of manual testing and automation testing:

**Manual Testing:**

1. Process: Manual testing involves human testers executing test cases manually without the use of automation tools. Testers interact with the software, validate its behavior, and identify defects.

2. Flexibility: Manual testing allows testers to adapt quickly to changes in requirements, user interfaces, or test scenarios. It is more suitable for exploratory testing and usability testing, where human intuition and observation play a crucial role.

3. Initial Setup: Manual testing does not require significant initial setup or investment in automation tools. Testers can start testing as soon as the application is available.

4. Maintenance: Manual tests require continuous manual execution, making maintenance efforts higher. As the application evolves or changes, test cases need to be updated and executed manually.

5. Skill Set: Manual testing relies on the expertise and experience of human testers to identify issues, provide feedback, and perform ad hoc testing based on their knowledge of the application.

6. User Experience: Manual testing allows testers to evaluate the user experience directly and identify issues related to usability, accessibility, and user satisfaction.

7. Exploratory Testing: Manual testing is well-suited for exploratory testing, where testers can freely explore the application, experiment with different inputs, and uncover unexpected defects or usability issues.

**Automation Testing:**

1. Process: Automation testing involves using automation tools and scripts to execute test cases and compare the actual results with the expected results automatically. Testers write scripts to automate repetitive and time-consuming tasks.

2. Speed and Efficiency: Automation testing can perform repetitive tests faster and more consistently than manual testing. It is suitable for regression testing, load testing, and performance testing, where a large number of test cases need to be executed repeatedly.

3. Reusability: Automation tests can be reused across different versions and builds of the software, saving time and effort in retesting. Test scripts can be executed repeatedly, reducing the overall testing cycle time.

4. Maintenance: Although automation tests require initial investment in script development and maintenance, once the scripts are created, they can be maintained and executed with less effort compared to manual testing.

5. Accuracy: Automation testing eliminates human errors and ensures consistent test execution. It can accurately compare expected and actual results, detect deviations, and report failures objectively.

6. Scalability: Automation testing allows for parallel execution of tests on multiple environments or configurations, making it suitable for testing across a wide range of platforms, devices, or browsers.

7. Performance Testing: Automation testing is particularly effective for load testing and performance testing, where simulations of high user loads can be generated and measured accurately.

It's important to note that manual testing and automation testing are not mutually exclusive. They can be used in combination based on project requirements and objectives. Manual testing is often preferred for early-stage testing, usability testing, and exploratory testing, while automation testing is useful for repetitive tasks, regression testing, and performance testing. The choice between manual testing and automation testing depends on factors such as project complexity, time constraints, budget, and the nature of the software being tested.