Unit-05

Software Testing and Software Evolution

5.1. Development Testing

Development testing is a critical phase in the software development lifecycle, aimed at identifying and resolving defects as early as possible. This phase includes various types of testing to ensure that the software functions correctly, meets requirements, and is free of bugs. Development testing is carried out during the development phase to identify and fix defects as they are introduced. This includes:

1) **Unit Testing**

→ To test individual components or modules of the software in isolation to ensure they work correctly by writing and executing test cases, fixing any defects found, and retesting to confirm the fixes. This helps ensure that each part of the software functions as expected.

2) **Integration Testing**

→ To test combined parts of an application to ensure they work together by planning integration tests, executing them, identifying any integration issues, and fixing and retesting these interactions. This ensures that different modules or services within an application communicate and function together seamlessly.

3) **System Testing**

→ To test the entire system as a whole to ensure it meets the specified requirements by developing comprehensive test cases that cover all functionalities, executing these tests to validate the system's overall behavior, logging any defects found, and fixing and retesting the entire system. This ensures that the complete software solution performs correctly and meets all specified requirements before release.

5.2. Test-Driven Development

Test-Driven Development (TDD) is a software development methodology in which tests are written before writing the actual code. The process of TDD follows a repetitive cycle of writing a test, writing the code to pass the test, and then refactoring the code. This approach ensures that the software development is driven by automated test cases

from the very beginning. TDD is a software development approach where tests are written before the code. The process involves:

1) **Writing a Test for a New Feature**

→ Define a test case for a new feature or functionality based on the requirements. This test case initially fails because the corresponding code has not yet been implemented. This step helps clarify the requirements and design before coding begins.

2) **Writing the Minimal Code to Pass the Test**

→ Develop the simplest possible code that can make the previously written test pass. The goal is to implement just enough functionality to satisfy the test case. This ensures that the feature works as intended and meets the specified requirements.

3) **Refactoring the Code While Ensuring All Tests Still Pass**

→ Improve and clean up the code without changing its behavior. Refactoring involves optimizing the code structure, improving readability, and removing any redundancies. Throughout this process, all existing tests, including the new one, must still pass to ensure that the functionality remains intact.

5.3. Release Testing

Release testing is a phase in the software development lifecycle where the software is tested in a production-like environment to ensure it meets all the specified requirements and is ready for deployment to end users. The primary goal of release testing is to validate the software's quality, performance, and stability before it is released to the market or delivered to the customer. Release testing ensures that the software is ready for deployment. It includes:

1) **Alpha Testing**

→ Alpha testing is conducted by internal teams to identify early-stage defects and ensure the software's functionality, usability, and performance in a controlled environment. It aims to validate critical functionalities before wider deployment, with feedback used to prioritize and resolve issues promptly.

2) **Beta Testing**

→ Beta testing involves releasing the software to a limited number of external users or customers to gather real-world feedback. This phase aims to validate the software's readiness for a broader release by identifying any remaining issues,

usability concerns, or unexpected behaviors that were not uncovered during alpha testing. Feedback collected from beta testers is crucial for making final adjustments and improvements before the official release.

5.4. User Testing

User testing, also known as usability testing, is a technique used in the software development process to evaluate a product by testing it with real users. The main objective is to ensure that the software is user-friendly and meets the needs and expectations of its intended users. This type of testing focuses on the user experience (UX) and helps identify usability issues before the product is launched. User testing involves real users testing the software in a real-world environment. This includes:

1) **Acceptance Testing**

→ This ensures the software meets the user's requirements and is ready for deployment. It involves validating that the software functions as expected and meets specified criteria before being released to users.

2) **Usability Testing**

→ This evaluates how easy and intuitive the software is to use from the perspective of end-users. It focuses on aspects such as navigation, user interface design, and overall user experience to identify any usability issues that may impact user satisfaction and efficiency.

5.5. Evolution Processes

The software evolution process involves making changes to software after its initial development to improve performance, add features, or fix defects. This process includes:

1) **Evolving Requirements:** Adjusting the software to meet changing user needs involves continuous communication with stakeholders to understand new functionalities or modifications required for better usability and market relevance.

2) **Maintenance Activities:** Fixing bugs and making improvements to keep the software functional and up-to-date are essential to ensure reliability, security, and compatibility with evolving technology standards. Regular

updates and patches help mitigate risks and enhance overall user experience.

5.6. Legacy Systems

Legacy systems are outdated software systems that are still in use. They may be critical to an organization's operations but can be challenging to maintain due to:

→ **Obsolete Technologies:** Using outdated technology that is no longer supported poses significant risks. Security vulnerabilities remain unpatched, exposing applications to breaches. Compatibility issues hinder performance and integration with modern software and hardware. Maintaining such technology may lead to higher costs and difficulties in finding skilled developers familiar with outdated systems, jeopardizing project longevity.

→ **Complexity:** Using outdated technology that's no longer supported brings risks. Security issues remain, leaving applications vulnerable. Compatibility problems arise with modern software and hardware, impacting performance. Maintaining it becomes costly, finding skilled developers is harder. Plus, it lacks documentation, making it complex to modify and prone to errors.

→ **Integration Issues:** Using outdated technology can lead to integration issues with newer systems and technologies. Compatibility challenges may arise, impacting performance and functionality. This can hinder the ability to leverage modern advancements and may require additional resources to resolve compatibility issues.

5.7. Software Maintenance

Software maintenance involves modifying and updating software after it has been delivered to correct faults, improve performance, or adapt to a changed environment. Types of maintenance include:

Corrective Maintenance

It involves fixing bugs and defects found in the software:

→ When users encounter issues like errors or crashes while using the software, corrective maintenance focuses on identifying and repairing these problems promptly.
→ It also includes conducting tests to ensure that fixing one issue does not unintentionally create new ones.

Adaptive Maintenance

It modifies the software to work in new or changing environments:

→ As technology evolves, the software may need adjustments to stay compatible with updated systems or new hardware.
→ Adaptive maintenance ensures that the software remains functional and efficient in different environments by making necessary changes without disrupting its core operations.

Perfective Maintenance

It enhances the software's performance and adds new features:

→ This type of maintenance aims to improve the software's speed, efficiency, and user experience.
→ It includes optimizing code to make the software run faster, adding new features that users request, and enhancing existing features based on feedback to make them more useful and intuitive.

Preventive Maintenance

It makes changes to prevent future problems:

→ Instead of waiting for issues to arise, preventive maintenance involves taking proactive measures to anticipate and address potential problems.
→ This can include regular updates to address security vulnerabilities, reviewing and refining code to improve its quality, and updating documentation to keep it clear and up-to-date for developers and users alike.