Software Engineering - Task 1 2024

Software Development Life Cycle (SDLC):

Q: Explain the different phases of SDLC (e.g., requirements analysis, design, implementation, testing, maintenance).

The software development life cycle is an efficient development process used to build high quality software. The goal of this process is to minimise the risks by planning ahead (Amazon, n.d.). The SDLC is divided into the following seven phases:

- Plan: In this phase, the project leader officially collects information like cost-benefit analysis, scheduling, resource estimation, and allocation (Amazon, n.d.). This is followed by discussions with the client about product requirements and expectations allowing the project leader to make a precise plan, success criteria, and create a set schedule.
- Design: During the "Design" phase engineers will brainstorm ideas and select the best approach to the problem identified during the planning phase. In this phase they will usually consider integrating pre-existing modules, make technology choices, and identify development tools (Amazon, n.d.).
- Implement: In the implementation phase, the team will start working on the solution identified during the design phase. They will follow the development schedule produced in the planning phase and will write out the code for the product.
- Test: During this phase, the team will conduct a series of automated and manual tests to identify, and hopefully, fix all the existing problems. This phase usually runs in parallel with the implementation phase (Amazon, n.d.).
- Deploy: When this phase starts, software teams make two copies of their code, one called "production" and the other called the "build environment" (Amazon, n.d.). "Production" is the customer copy of the code. Having two copies of the code has multiple benefits, most prominent being allowing continued development on the "build environment" without the need for customers to stop using it as they are on the "production copy."
- Maintain: This is probably the longest phase lasting as long as the product is supported. In this phase, the team fixes any bugs that are reported, provides customer support, implements any new features and creates new releases.

- Evaluation: This phase consists of post-implementation review and lessons learned. In this phase, the team will evaluate the project’s success against the criteria created during the planning phase. The team will also document the lessons learnt. This allows them to use best practices for future projects.

Q: Discuss why following an SDLC is essential for successful software development.

Following an SDLC is essential for successful software development as it helps in many areas like, providing aids in project planning, making project tracking easier, faster development. This leads to improved client relations, decreased project risks, and decreased project management expenses and the overall cost of production (Synopsys, n.d.).

Programming Paradigms:

Q: Describe imperative, declarative, and object-oriented programming paradigms.

Imperative Programming

Imperative programming focuses on how to achieve a result, giving step-by-step instructions to the computer (guide, 2018). It often uses mutable variables, meaning data can be changed throughout the program. C, Fortran, and Basic are examples of imperative languages (Rani, 2024). Imperative code is often not reusable because the context behind it changes, leading to developers needing to change some or all of the code to reuse it.

Declarative Programming

Declarative programming focuses on describing what needs to be done rather than how. The programmer specifies the desired outcome, and the system figures out how to achieve it. Declarative languages often use immutable variables, where data remains constant. Common examples of declarative programming languages include SQL, HTML, XML, and CSS (Rani, 2024).

Object-Oriented Programming (OOPS)

Object-oriented programming (OOPS) builds on top of the imperative paradigm. It organises code around objects, which bundle attributes and the methods. This approach promotes modularity and reusability of code. Modern languages like C++, Java, and Python all heavily utilise OOPS concepts (Rani, 2024).

Q: Explain how Python supports these paradigms.

Python was designed to be an expressive (Pramanick, 2023), readable, and simple language (Python.org, n.d.). Thus, the focus of Python is on the Imperative and Object-Oriented programming paradigms. In terms of imperative programming, Python focuses on step-by-step functions that are “interpreted” at runtime. For example, in Python, if we are writing a program that adds a list of integers, we will use a for loop to loop through the integers of the list, adding them one by one. This is following the imperative approach as we are giving the computer specific, step-by-step directions. Python also offers the ability to create classes and objects, allowing it to embrace object-oriented principles. For example, in Python, we can define a class called Animal, where there are variables like name and type, and functions like eat, sleep and drink. After this, we can, using ‘inheritance’, an important Object Oriented Programming principal, ‘inherit’ the class into a class called Dog. The Dog class will ‘inherit’ the base variables (name and type) and the base functions (eat, sleep and drink), but will also be open to extension. From here, we could add more variables, like breed, and define Dog-specific functions like bark. Lastly, even though Python does not focus on the declarative programming paradigm, it facilitates the declarative programming paradigm by providing predefined functions to execute programs. The function sum(), which allows you to add a list of integers, is an example, as you are ‘declaring’ what you want to do rather than writing it out in machine instructions.

Software Testing:

Q: Define unit testing, integration testing, and acceptance testing.

Unit Testing

A unit test is a block of code that runs test cases on other blocks of code (Amazon, n.d.). This usually runs tests on only one function at a time. It works by giving a function the required arguments and checking it against expected output. For example, in case of a function that adds two integers, the unit test's inputs will be 4 and 5 and expected output will be 9.

Integration Testing

Integration testing focuses on how independent units of code collaborate (Sanjoy, 2023). In this, testers combine multiple units to ensure proper data exchange and combine multiple units to ensure proper data exchange and communication between them. This helps find any discrepancies in how the units work together. Big Bang integration testing is a common example of integration testing. An example of an integration test is when testing the interaction between the shopping cart, payment gateway, and order confirmation modules. We can set up an integration test to do the following actions:

- Add a product to the shopping cart and proceed to checkout.
- Enter valid credit card information and submit the payment.

After this, we can verify the following assertions:

- Verify the shopping cart total is correctly reflected in the payment gateway.
- Verify the payment gateway successfully authorises the transaction.
- Verify a new order is created with the purchased product details.
- Verify an order confirmation email is sent to the customer with the order details.

This sample test ensures data flows smoothly between the shopping cart, payment gateway, and order confirmation modules. It verifies successful product selection, payment authorization, order creation, and email notification.

Acceptance Testing

Acceptance testing is the last level of testing. In this, the testers simulate real-world use by the target audience to validate if the entire system fulfils the intended requirements The two common forms of acceptance testing are User Acceptance Testing (UAT) and Business Acceptance Testing (BAT) (Pankaj, 2024). An example of an acceptance test is when testing an e-commerce website. We could design an acceptance test in the following way:

- **Scenario:** User adds a product to the cart
- **Steps:**
    - User navigates to the product page of a desired item.
    - User selects a quantity (e.g., using a dropdown menu or quantity box).

    - User clicks the "Add to Cart" button.
- **Expected Result:**
    - The product and its chosen quantity are displayed in the shopping cart.
    - The total price in the cart reflects the added product and its quantity.
    - The user receives a confirmation message (visually or audibly) indicating successful addition to the cart.
- **Pass/Fail Criteria:**
    - Pass: If all the expected results occur as described.
    - Fail: If any of the expected results are missing or incorrect (e.g., product not added, quantity wrong, price mismatch, no confirmation message).

This acceptance test verifies a core functionality of the e-commerce website from the user's perspective. By defining clear steps, expected results, and pass/fail criteria, it ensures the system meets a crucial user requirement.

In conclusion, the three main types of software tests, unit tests, integration tests, and acceptance tests. These tests are increasingly complex due to the amount of code they are testing, with unit tests being the simplest and integration tests being the most complex.

Q: Discuss the importance of testing in software development.

Testing is a vital component of the software development lifecycle. It allows early detection of issues through systematic testing (Baweja, 2023). It also minimises waste of time and resources as bugs can be detected and fixed early. Different types of tests are used at different stages, for example unit tests are used at a lower level and are used frequently, while acceptance testing is done at the end of the development period as it requires human intervention, thus taking more time.

Version Control:

Q: Explain the purpose of version control systems (e.g., Git).

Version control systems are a necessary part of managing software development projects, allowing users to record changes made to files by keeping a track of modifications done in the code (Soumya, 2022). The version control system, for example Git, meticulously tracks every modification a user makes to a project. This allows developers to revert to previous versions if necessary, because of an unfixable bug, safeguarding the project. Furthermore, VCS facilitates collaboration between developers allowing them to work on the same project simultaneously. The software then allows you to merge changes and identify, and fix, any conflicts that may arise. By providing such facilities, VCS foster seamless collaboration and are instrumental in maintaining project integrity and promoting efficient software development.

Q: Describe how version control helps manage code changes.

Version control systems (VCS) offer a robust solution for managing code changes throughout the SDLC. These work by meticulously tracking every change made to the codebase. This record allows developers to, as mentioned previously, revert to past versions, safeguarding their code against accidents. Furthermore, VCS allows multiple developers to effectively contribute to the same repository. Most importantly, VCS allows users to efficiently merge their changes, identifying and fixing any conflicts that may arise. By providing a clean interface to commit, revert and merge changes, VCS play a critical role in managing code changes (Soumya, 2022).

Bibliography:

Amazon. (n.d.). *What is SDLC? - Software Development Lifecycle Explained*. AWS. Retrieved March 15, 2024, from https://aws.amazon.com/what-is/sdlc/

Amazon. (n.d.). *What is Unit Testing? - Unit Testing Explained*. AWS. Retrieved March 18, 2024, from https://aws.amazon.com/what-is/unit-testing/

Baweja, M. (2023, November 2). *Why is software testing Important? | Engati*. Engati chatbot. Retrieved March 18, 2024, from https://www.engati.com/blog/importance-of-software-testing

guide, s. (2018, November). *What is imperative programming ? | Definition from TechTarget*. TechTarget. Retrieved March 17, 2024, from https://www.techtarget.com/whatis/definition/imperative-programming

Pankaj. (2024, January 8). *Acceptance Testing - Software Testing*. GeeksforGeeks. Retrieved March 18, 2024, from https://www.geeksforgeeks.org/acceptance-testing-software-testing/

Pramanick, S. (2023, December 26). *History of Python*. GeeksforGeeks. Retrieved March 17, 2024, from https://www.geeksforgeeks.org/history-of-python/

Python.org. (n.d.). *General Python FAQ — Python 3.12.2 documentation*. Python Docs. Retrieved March 17, 2024, from https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place

Rani, B. (2024, March 14). *Introduction of Programming Paradigms*. GeeksforGeeks. Retrieved March 17, 2024, from https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

Sanjoy. (2023, November 21). *Integration Testing - Software Engineering*. GeeksforGeeks. Retrieved March 18, 2024, from https://www.geeksforgeeks.org/software-engineering-integration-testing/

Soumya. (2022, June 29). *Version Control Systems*. GeeksforGeeks. Retrieved March 22, 2024, from https://www.geeksforgeeks.org/version-control-systems/

Synopsys. (n.d.). *What Is the Software Development Life Cycle (SDLC) and How Does It Work?* Synopsys. Retrieved March 17, 2024, from https://www.synopsys.com/glossary/what-is-sdlc.html