**Coding standards and document generator**

Good software development organizations want their programmers to maintain to some well-defined and standard coding style called coding standards.

They usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop.

The programmers need to maintain the coding standards otherwise the code will be rejected during code review.

**Purpose of Having Coding Standards**

The following are the purpose of having Coding Standards:

* A coding standard gives a uniform appearance to the codes written by different engineers.
* It improves the readability and maintainability of the code and it reduces complexity also.
* It helps in code reuse and helps to detect errors easily.

**Write a Function to Check if a Number is Even**

def a(x):return x%2==0

What does this function do?Can you guess what a(x) means?Is it easy to read or understand?

def is\_even(number: int) -> bool:

"""

Checks if a given number is even.

Returns True if even, False otherwise.

"""

return number % 2 == 0

Some of the coding standards are given below:

* **Limited use of globals:** These rules tell about which types of data that can be declared global and the data that can’t be.
* **Standard headers for different modules:** For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:
* Name of the module
* Date of module creation
* Author of the module
* Modification history
* Synopsis of the module about what the module does

Different functions supported in the module along with their input output parameters

Global variables accessed or modified by the module

* **Naming conventions for local variables, global variables, constants and functions:** Some of the naming conventions are given below:

Meaningful and understandable variables name helps anyone to understand the reason of using it.

Local variables should be named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names should start with a capital letter (e.g. **GlobalData**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).

It is better to avoid the use of digits in variable names.

The names of the function should be written in camel case starting with small letters.

The name of the function must describe the reason of using the function clearly and briefly.

* **Indentation:** Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:

There must be a space after giving a comma between two function arguments.

Each nested block should be properly indented and spaced.

Proper Indentation should be there at the beginning and at the end of each block in the program.

All braces should start from a new line and the code following the end of braces also start from a new line.

* **Error return values and exception handling conventions:** All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging.
* **Code should be well documented:** The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.
* **Length of functions should not be very large:** Lengthy functions are very difficult to understand. That’s why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.
* **Try not to use GOTO statement:** GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

**Literate programming and Software documentation; Documentation generators, Javadoc, phpDocumentor**

1. Literate programming" is a programming paradigm introduced by Donald Knuth in 1984. The idea behind literate programming is to treat programming as a form of literature, where the focus is on explaining the logic and design of the program in a human-readable way rather than just writing code for the computer to execute.
2. In literate programming, the source code is interspersed with natural language explanations, diagrams, and other forms of documentation to make the code more understandable and maintainable. Instead of writing code in a linear fashion, programmers write the code in the order that makes the most sense from a conceptual standpoint, with explanations and documentation throughout.
3. The literate programming approach encourages clear and expressive documentation of the design rationale, algorithmic choices, and overall structure of the program. This can make it easier for other developers to understand and modify the code, as well as for the original author to revisit and revise the code in the future.
4. Software documentation in the context of literate programming often goes beyond traditional documentation practices. It involves creating a cohesive narrative around the codebase, explaining not only what the code does but also why it does it in a particular way. This can include detailed explanations of algorithms, design patterns, and architectural decisions, as well as examples, usage scenarios, and other contextual information.

Overall, literate programming and software documentation work hand in hand to create more understandable, maintainable, and extensible software projects. By treating code as a form of literature and providing rich, human-readable explanations alongside the code itself, literate programming can enhance the quality and accessibility of software documentation.

**Documentation generators**

**What is a Documentation Generator?**

A **documentation generator** is a tool that **automatically creates documentation** (like manuals, API references, or function descriptions) **from your code**—especially from **comments or docstrings**.

They analyze the code and the comments (usually in a specific format) within the code to produce comprehensive documentation.

Two popular documentation generators are Javadoc for Java and phpDocumentor for PHP. Here’s an overview of each, along with examples.

Javadoc

**Javadoc** is a documentation generator for Java that creates HTML documentation from Java source code. The documentation is generated from special comments in the code, known as Javadoc comments. **Key Javadoc Tags**

| **Tag** | **Purpose** |
| --- | --- |
| @author | Author of the code |
| @version | Version of the class |
| @param | Describes a method parameter |
| @return | Describes the return value |

To generate Javadoc documentation, use the javadoc tool:

javadoc -d doc Calculator.java This command generates the documentation in the doc directory.

**It creates .html files for your class, with proper sections:**

* **Description**
* **Method Summary**
* **Parameters**
* **Return values**

**Open Calculator.html in your browser to view your documentation.**

**Use of Javadoc**

* Makes your code self-explanatory
* Helps other developers use your classes/methods
* Industry-standard for Java documentation
* Required in many professional Java projects

Javadoc Example

Here's an example of how to use Javadoc in a Java class:

/\*\*

\* The `Calculator` class provides methods to perform

\* basic arithmetic operations such as addition, subtraction,

\* multiplication, and division.

\* \* @author Melvin

\* @version 1.0

\*/

public class Calculator {

/\*\*

\* Adds two integers and returns the result.

\* \* @param a the first integer

\* @param b the second integer

\* @return the sum of `a` and `b`

\*/

public int add(int a, int b) {

return a + b;}

/\*\*

\* Subtracts the second integer from the first and returns the result.

\*

\* @param a the first integer

\* @param b the second integer

\* @return the difference of `a` and `b`

\*/

public int subtract(int a, int b) {

return a - b;

}

// Other methods with similar Javadoc comments...

}

phpDocumentor

**phpDocumentor** is a documentation generator for PHP. It parses PHP source code and PHPDoc comments to produce documentation in various formats, including HTML and PDF.

phpDocumentor Example

Here's an example of how to use phpDocumentor in a PHP class:

<?php

/\*\*

\* The Calculator class provides methods to perform

\* basic arithmetic operations such as addition, subtraction,

\* multiplication, and division.

\*

\* @package MyLibrary

\*/

class Calculator {

/\*\*

\* Adds two numbers and returns the result.

\*

\* @param float $a The first number.

\* @param float $b The second number.

\* @return float The sum of $a and $b.

\*/

public function add($a, $b) {

return $a + $b;

}

/\*\*

\* Subtracts the second number from the first and returns the result.

\*

\* @param float $a The first number.

\* @param float $b The second number.

\* @return float The difference of $a and $b.

\*/

public function subtract($a, $b) {

return $a - $b;

}

// Other methods with similar PHPDoc comments...

} To generate documentation with phpDocumentor, use the following command:

phpdoc -d . -t docs

This command scans the current directory (.) for PHP files and generates documentation in the docs directory.

Summary

Both Javadoc and phpDocumentor help in creating comprehensive, standardized documentation directly from the source code by utilizing special comments. This approach ensures that the documentation is closely tied to the code and is easier to keep up to date.

Benefits of Using Documentation Generators:

**Consistency**: Ensures documentation is consistently formatted.

**Up-to-date**: Encourages keeping the documentation in sync with the code.

**Accessibility**: Makes it easier for developers to understand and use code written by others.

By leveraging these tools, developers can maintain high-quality documentation with minimal effort, enhancing both internal and external communication regarding the software project.

Git and testing process

1. Configure Git with your name and email address:

2. Create a Git repository named “Experiment” in the folder C:/exam:

cd C:/exam

mkdir Experiment

cd Experiment

git init

Explanation:

cd C:/exam – Changes to the C:/exam directory (using /d is important when switching drives in Command Prompt).

mkdir Experiment – Creates a folder named Experiment.

cd Experiment – Moves into the Experiment folder.

git init – Initializes a new Git repository inside the folder.

### i) Create two files in a Git repository

cd C:/exam/Experiment # Go to your Git repo folder

echo "Hello" > file1.txt # Create first file

echo "World" > file2.txt # Create second file

You can confirm the files are created with:

ls

### ii) Commit with and without moving to the staging area

#### 🔹 **Commit WITH moving to the staging area (standard method):**

git add file1.txt file2.txt # Stage both files

git commit -m "Added file1 and file2 with staging"

#### 🔹 **Commit WITHOUT moving to the staging area (skip** git add**):**

Use the -a (all) option with git commit. It **only works for modified or deleted tracked files**, not new ones.

So for **new files**, you cannot skip staging. But for example, if you modify a file later:

echo "New line" >> file1.txt # Modify file1

git commit -am "Modified file1 without separate staging

| **Task** | **Command** |
| --- | --- |
| Create files | echo "Hello" > file1.txt |
| Stage files | git add file1.txt file2.txt |
| Commit with staging | git commit -m "message" |
| Commit without staging (modifications only) | git commit -am "message" |

**Quality Assurance (QA) or Software Quality Assurance (SQA)** is a term used to describe a systematic process of assessing the quality of software solutions and implementing ways to improve them.

In a broader context, SQA refers to the organizational part of quality management when everyone on the team knows exactly in what order things need to be done and strictly adheres to these rules. This is the basis without which further testing is impossible.

**Processes Involved**

* Recognize and identify quality standards applicable in software development.
* Conduct regular quality reviews.
* Carry out processes for recording test data.
* Arrange and maintain artifacts for QC measures.

**Examples of Quality Standards:**

* **ISO/IEC 25010:** A quality model that defines various characteristics of software quality, including functional suitability, performance efficiency, security, and more.
* **CMMI (Capability Maturity Model Integration):** A process improvement approach that helps organizations improve their software development processes.
* **IEEE Standards:** The Institute of Electrical and Electronics Engineers (IEEE) has various standards for software engineering, such as IEEE 829 for test documentation.

Any 3 examples of quality stds

**Testing** is the process of checking a product as a whole and each of its parts separately. It can be carried out in many different ways using both manual approach and automated testing tools depending on the scale and goals of a project.

**Steps Involved**

1. Develop a strategy outlining what aspects of the product will be tested, how testing will be conducted, and what criteria will define success.
2. Create test cases that cover a wide range of functionalities of a product.
3. Run test cases, either manually or through automated tools.
4. Document any issues or deviations from expected behaviour.Repeatedly test the software after each change or update to ensure that new modifications haven’t affected the product’s performance and functionality.

example

The **whole product** (e.g., the food delivery app) works as expected.**Each part** of the product (e.g., login page, restaurant list, payment, order tracking) works correctly **individually**.

**Create Test Cases**

Each feature will have test cases. For example:

**Feature:** Add food item to cart  
**Test case:**

* Step 1: Select a restaurant
* Step 2: Click on a food item
* Step 3: Click "Add to Cart"
* **Expected result:** Item should be visible in the cart

#### **. Run Test Cases (Manually or Automated)**

* **Manual test:** A tester follows the above steps manually on a mobile device.
* **Automated test:** A script runs the above steps using Selenium/Appium.

Here are the two types of software testing conducted for this purpose:

* **Functional,** which focuses on the product’s functionality from the user’s point of view, e.g. its ease of use, convenience, UI design, absence of glitches, clear contact forms, price-to-quality ratio, and so on.
* **Non-functional (or structural)**, performed to assess the inner components of the software, e.g. its code maintainability, scalability to handle increased loads, security measures, compatibility across different platforms and browsers, the list goes on.

Of course, assessing the quality of a product through structural testing without having expertise in this field isn’t easy. Therefore, structural quality testing is entrusted to experienced testers who possess a deep understanding of software architecture, coding principles, and system internals. Functional testing, on the other hand, is performed through software quality management activities, such as QA, QC, and testing.

**Methods Used in QA**

Let’s dive into the key QA methods that teams rely on to check their websites and applications for issues and bugs. In general, there are two types — non-functional and functional.

Let’s start by quickly going through the phases of **non-functional testing**, which look like this:

* **Vulnerability testing** aims to detect any potential issues that could expose a website or application to security threats such as cyberattacks, data breaches, or unauthorized access.
* **Compatibility testing** is conducted to check the product’s compatibility with operating systems, browsers, hardware, and software to ensure that it functions correctly and consistently across different environments.
* **Usability testing** is executed to ensure that the product’s user interface is easy to understand and use so that end users can interact with a website or application intuitively and efficiently.
* **Performance testing** allows teams to ensure that the product’s performance remains consistent and flawless under various conditions, including high loads, low battery, and so on.

Now, it’s time to look more closely at **functional testing** methods. Here they are:

* **Unit testing**  is used to test different parts and components of the software in isolation. Unit tests are written before building a module to test specific functions, classes, or modules in the codebase.
* **Integration testing** focuses on evaluating how different components or modules of the software interact with one another when integrated. It verifies that data flows smoothly between these integrated parts and that they work cohesively as a whole.
* **System testing** extends to the whole framework, enabling test engineers and developers to identify stability issues and spot bugs they haven’t noticed before.
* **Acceptance testing** is typically one of the final phases before the product’s release, which involves end-users and stakeholders to validate that the product meets requirements and expectations.

 application

**E-commerce Website Development**

**Project Overview:** An e-commerce company is developing a new website to allow customers to purchase products online. The project involves several features, including user registration, product browsing, a shopping cart, and payment processing.

**QA Process:**

1. **Requirement Analysis:**

* QA team reviews the project requirements to understand the functionalities and performance criteria of the website.
* Identify any potential issues or ambiguities in the requirements.

1. **Test Planning:**

The QA team creates a test plan, outlining the testing strategy, resources, schedule, and scope.

Decide on testing types (e.g., unit testing, integration testing, user acceptance testing).

1. **Test Case Development:**

QA team writes detailed test cases for each feature of the website. For example:

* + - **Test Case 1:** Verify that a user can successfully register with a valid email and password.
    - **Test Case 2:** Verify that the shopping cart correctly adds, removes, and updates products.
    - **Test Case 3:** Ensure that the payment gateway processes transactions securely and accurately.

1. **Environment Setup:**

Set up a testing environment that mirrors the production environment as closely as possible.

This includes the same server configurations, databases, and third-party integrations.

1. **Test Execution:**

QA team executes the test cases in the testing environment.

Log any defects or issues discovered during testing.

* + - **Example Defect:** The shopping cart fails to update the total price when a product is removed.

1. **Defect Reporting and Tracking:**

Report defects to the development team using a defect tracking system (e.g., JIRA).

Prioritize defects based on their severity and impact on the project.

Example: The payment gateway issue would be considered a critical defect, while a typo on the homepage might be minor.

1. **Regression Testing:**

After defects are fixed, regression testing ensures that the fixes do not introduce new issues.

QA team re-tests the affected areas and related functionalities.

1. **User Acceptance Testing (UAT):**

Involve end-users or stakeholders to test the software in a real-world scenario.

Gather feedback to ensure that the product meets their expectations.

By following a thorough QA process, the e-commerce website is released with minimal defects, providing a smooth and reliable shopping experience for users. This reduces the risk of negative customer feedback, cart abandonment, or loss of revenue due to software issues.

Expected questions

SHORT ANSWER TYPE

1. Any two coding standards.
2. literate programming in software development.
3. how you can view the commit history along with any three options.
4. How branching in Git.
5. How GIT COMMIT” different from “GIT PUSH”?
6. What is branching in Git
7. Explain the purpose of coding standards and guidelines in software developing industries.
8. Differentiate between software documentation and literate programming.
9. What is documentation generators and any 2 of this..
10. Explain the features and use of the Javadoc tool in coding.
11. Explain the need for the PHPdoc tool in coding.
12. Explain the purpose of the version control system.
13. Compare distributed version control systems over centralized version control system.
14. Explain how to modify a file in a git repository.
15. Your colleague has created a Git repository for a project and shared the repository URL with you. How would you clone this repository to your local machine?
16. You’ve been working on a file in your project and want to check which files have been modified before committing your changes. How would you do this?
17. You’re working on a project where multiple team members are pushing changes to the remote repository. Before starting your work, you want to ensure you have the latest updates. How would you pull the latest changes from the remote repository?

Long answer type

1. core operations of Git with the help of commands.
2. how do you create, switch and view branches in Git? Also explain how to merge commits
3. HOW TO DO THE FOLLOWING:
   1. Configure git with your name and email address.
   2. Create a git repository named “Experiment” in the folder C:/exam

4.HOW TO DO THE FOLLOWING:

* + 1. Create two files in a git repository
    2. commit with and without moving to the staging area.

5.Different types of merging in git with example