How do you review code?

Code reviews help to enable faster and cheaper completion of software development projects. Code reviews can help software development teams in achieving software quality, security, reliability, robustness, and efficiency. That is why code reviews should be dealt with utmost priority and the best practices for code reviews should be maintained throughout the development process.

My code review process includes these steps:

Goals and expectations:

I believe it is very important to set goals and expectations before doing code reviews. Goals include enforcing acceptable coding standards in the company.

Checklist:

I create a checklist of things to look for to make sure nothing is missed out. In a code review, I always check key things such as Structure, Logic, Style, Performance, Test Coverage, Design, Readability, Maintainability, Functionality, Scalability, Reusability, and Security.

Break into short tasks:

I break code reviews into short tasks which can take up to 30 minutes to 1 hour to accomplish. It makes the process more agile. I do not review code for more than 60 minutes to avoid mistakes.

Giving useful feedback:

I try to give precise comments and feedbacks that can help the developer understand the code error. I encourage developers and try to build a positive culture.

Commit often: I don't wait days for committing, rather I commit often to make the process faster.

Automating the process: Before going into the manual review, I first test and build. Automated checks save time and make the code review process faster.

How do you enforce coding standards?

Coding standards help to make a code readable, maintainable, and reusable. Also, it increases efficiency by improving application performance. Moreover, bug identification and fixing could be easier.

The following points can help to enforce coding standards:

Defining the standard:

Choosing the right coding standard is important. The right coding standard should offer the following things:

Limited use of globals

Standard headers from different modules

Defining naming conventions for local variables, global variables, constants, and functions

Proper indentations

Error return values and exception handling conventions

Avoid using a coding style that is too difficult to understand

Avoid using an identifier for multiple purposes

Code should be well documented

Length of functions should not be very large

Adopting the standard:

Static analyzers can help in checking codes against the rules from the selected coding standards automatically. There are many tools available in the market that can help us adopt the standard. For example, there are command-line tools that can be integrated with build tools like Ant or Maven. Also, we can use these tools directly from our IDE so that developers see the errors as they type and solve them immediately.

Enforcing the standard:

To enforce the standard, we can integrate static analyzers into our Continuous Integration (CI) builds. For example, it can be run every night to determine how many violations exist in the code.

Training:

Developers should understand the importance of coding standards. Therefore, proper training about coding standards should be organized.

How do you plan what kind of approach you take for test automation - what libraries to use, how does it work in couple of years, how to make it easy to maintain, etc? What are the main points to consider?

Requirement Analysis:

In this step, we analyze what we require from the project. We need to involve all the key stakeholders in gathering business requirements. After gathering all the requirements, we should define our goal for the project.

Risk Assessment:

We should know about the potential risks involved in running automated tests and what kind of impact it can have by implementing a Risk Assessment.

Planning:

We should establish a plan before test automation. Such as which test cases should be automated and when the test automation process should start. A good strategy for Test Automation is to start it from a very early stage. In this way, we can make the process speedier and make it cost-effective. Usually, we automate tasks that carry high risk, are highly repetitive, require testing over multiple browsers, operate through multiple data sets, and can be tested with clear pass/fail. Also, tasks that can take a lot of time if manual testing is implemented, should be automated.

Selecting frameworks and testing tools:

We should select frameworks and tools that can help us to achieve our goals and fulfill the requirements of the project. While selecting tools we need to think about the future, does the tool fulfill our future requirements? Also, the tools need to produce robust tests and should be adaptive for most OS, languages, and platforms.

Execution:

We need to provide faster feedback to the development team. Therefore, the execution plan should include day-to-day tasks and procedures. Tools like Bamboo or Jenkins can help in this regard.

Result and Analysis:

We need to identify which test cases have consumed more time and should optimize them. Also, we need to discuss with testers, developers, and everyone involved in the process for reviewing our test automation strategy.

Documentation:

This is a very important part. We need to document everything such as analysis, results, and suggestions.

Code testability, how do you enforce it?

"Testability is the degree to which software can be tested completely and efficiently". Testability is very important to get a clear picture of a product’s limitations. It helps in making crucial decisions that can help the business win its reputation. A testable product means it is maintainable and operable. In other words, Testability is a way of ensuring quality.

The following things can help to make the code testable:

- Units of code are short
- Data models separation
- Units do a single or fewer tasks
- Dependencies are not hardcoded
- Adhere to the SOLID principles

To enforce testability following things can help:

Shifting left: To enforce testability my first approach will be implementing Shift-Left Testing, meaning that, starting the testing process early in the development lifecycle. By shifting left in the testing process companies can accommodate releases that happen faster, usually on a weekly or daily basis.

Training:

It is important to understand the importance of code testability to deliver a high-quality product. We should provide training to developers and testers so that they realize the value of testability.

Planning:

Before starting the development process, there should be a precise plan about the Testing Lifecycle. It helps testers and developers understand the objectives, tasks, and outcome of the project.

Integration:

To understand where and at which stages testing should happen, all development, project management, and operations processes should be integrated with Testing.

Defining Quality Standards:

The quality standards should be defined so that there is a sync between development and quality assurance and identify any deviations from the expected outcome.

Checklist:

The Testing team should have a clear idea about what the application is expected to do. Also, they should know how the application reacts to abnormal scenarios. Checklists should facilitate addressing issues on how to create different scenarios.

Design:

The code should be designed in a way that ensures the product is testable under different scenarios.

How do you make sure that the product is testable?

To check whether the product is testable or not I'll perform smoke testing. By smoke testing, I'll check whether the deployed software build is stable or not. In simple terms, I'll verify that the essential features are working.