SE 2142

Week 8-9

7 STEPS TO IMPROVE CODE QUALITY

1. Adopt a coding standard

A coding standard, a set of guidelines for writing code, ensures consistency and readability. It helps to standardize the code's format, structure and style, making it easier for other developers to understand and maintain.

The main benefits of adopting a coding standard include:

- **Consistency:** A coding standard ensures that all code is written consistently, reducing the risk of bugs and improving the overall reliability of the code
- **Readability:** Well-written code is easy to read and understand. Coding standards ensure code is easy to read and maintain, making it easier for other developers to pick up where you left off
- Maintainability: A coding standard helps to ensure that code is written in a way that is easy to maintain over time. This can reduce the time and effort required to make changes to the code in the future.
- **Improved teamwork:** When everyone on a team follows the same coding standard, it makes it easier for developers to work together effectively. It helps to prevent misunderstandings and conflicts between team members.

Many established coding standards are available, and choosing the right one for your project will depend on your project's programming language and specific needs.

For example, PEP 8 is a widely-used coding standard for Python, while the Google Java Style Guide is a popular choice for Java.

2. Write automated tests

Automated tests are a way to catch bugs early on and prevent them from becoming bigger problems. Automated tests also ensure that changes to the code do not break existing functionality, providing confidence when making changes to the codebase.

The benefits of writing automated tests include:

- **Early bug detection:** Anything that helps you catch bugs early on saves time and effort in the long run and improves the overall quality of the code. Catching bugs later on is always more challenging to fix
- **Improved confidence when making changes:** When you have automated tests in place, you can confidently make changes to the code, knowing that the tests will catch any bugs you might have introduced
- **Increased speed and efficiency:** Automated tests can be run quickly and efficiently, making it easier to catch bugs and verify that the code works as expected.
- **Reduced manual testing effort:** Replacing manual testing efforts whenever you can frees up time and resources for tackling other tasks.

When writing tests, covering all relevant scenarios and edge cases is essential. This will help ensure the code is thoroughly tested, and bugs are caught early on. It’s also a good idea to write tests before you write the code, as this can help to clarify the requirements and ensure that the code is written to meet those requirements.

3. Use version control

Version control tracks change to the code over time, making it easier to revert to previous versions if necessary. Version control also enables multiple developers to work on the same codebase simultaneously without causing conflicts or losing work.

The benefits of using version control include:

- **Track changes over time:** Providing a way to track changes to the code makes it easier to revert to previous versions if necessary. This can be especially useful in case of bugs or other issues with the code
- **Collaboration:** By enabling multiple developers to work on the same codebase simultaneously without causing conflicts or losing work improves teamwork and improves the efficiency of developers working together
- **Backup:** Version control provides a backup of the codebase, making it possible to recover from accidents or mistakes
- **Improved transparency:** A clear history of changes to the code makes it easier to understand who made changes and why. This can improve transparency and accountability within a team

Many version control systems are available, with Git being one of the most popular. When using version control, it’s essential to follow best practices, such as using branches, writing descriptive commit messages, and regularly merging changes into the main codebase.

4. Refactor your code regularly

Refactoring is changing the structure of the code without changing its behavior. It can involve rewriting parts of the code to make it more readable, efficient or maintainable. The benefits of refactoring code include:

- **Improved readability:** Makes code more readable and straightforward for other developers to understand and maintain
- **Increased efficiency:** Makes code more efficient, reducing the time and resources required to run it
- **Enhanced maintainability:** Makes code more maintainable, reducing the time and effort required to make changes to the code in the future
- **Reduced risk of bugs:** Improving the overall reliability of the code
-

When refactoring code, it’s essential to keep the following in mind:

- **Write automated tests:** Before refactoring code, it’s essential to write automated tests to ensure that the code is working as expected. This will help to catch any bugs introduced during the refactoring process.
- **Make small, incremental changes:** Small changes are essential when refactoring code, as significant changes make catching bugs problematic. Small changes also make it simple to revert to previous versions when necessary.
- **Document changes:** It’s a good idea to document changes made during the refactoring process so that other developers can understand why changes were made and how the code works

5. Use code reviews

Code reviews are a process where one or more developers review the code written by another developer, providing feedback and suggestions for improvement. A must for all successful app development teams, code reviews help to catch bugs, improve code readability and ensure that code meets standards and best practices.

The benefits of using code reviews include the following:

- **Improved code quality:** Catches bugs and improves the overall quality of the code
- **Enhanced collaboration:** Enables developers to work together and collaborate, improving teamwork and communication
- **Increased knowledge sharing:** Provides a way to share knowledge and best practices within a team, helping to improve the overall quality of the code
- **Reduced risk of bugs:** Catching bugs early on improves the overall reliability of the code

When conducting code reviews, it’s essential to keep the following in mind:

- **Encourage open and honest feedback:** Code reviews should be a safe space where developers can provide honest and constructive feedback to one another
- **Focus on the code, not the person:** Code reviews should focus on the code itself, not the person who wrote it. Feedback should be constructive and focused on improving the code rather than criticizing the individual
- Make it a team effort: Code reviews should be a team effort, with multiple developers providing feedback and working together to improve the code

6. Use a linter

A linter is a program that checks the code for syntax errors, style violations and possible bugs. Using a linter, developers can quickly and easily identify problems in their code and make necessary improvements.

The benefits of using a linter include:

- **Improved code quality:** Helps to catch potential issues in the code, reducing the risk of bugs and improving the overall quality of the code
- **Consistent code style:** Helps to enforce a consistent code style, making code easier to read and maintain
- **Enhanced productivity:** Makes the development process more efficient by quickly and easily identifying potential issues in advance
- **Reduced errors:** Helps reduce errors in the code and improve the overall code reliability

When using a linter, it’s essential to keep the following in mind:

- **Choose a linter that is appropriate for your language:** There are many different linters available, and it’s essential to choose one that is appropriate for your language
- **Configure the linter to meet your needs:** You can configure linters to meet specific needs such as enforcing a particular coding style or checking for specific issues
- **Integrate the linter into your development process:** Linters can be integrated into your development process so that they are run automatically whenever changes are made to the code

By catching potential issues in the code, enforcing a consistent code style and making the development process more efficient, linters can help ensure high-quality code is easy to maintain.

7. Collaborate with other developers

Collaborating with other developers is an essential step in improving code quality. Collaboration brings various perspectives to the development process, helping to catch potential issues and improving the overall quality of the code. Particularly important when creating marketing assets that are a part of the customer journey, collaboration ensures apps are well-integrated with websites and other assets, providing a seamless and high-quality customer experience.

The benefits of collaborating with other developers include the following:

- **Improved code quality:** Bringing various perspectives into the development process helps to catch potential issues and improve the overall code quality
- **Enhanced knowledge sharing:** Spreads knowledge and best practices and improves the overall skill level of the development team
- **Reduced risk of bugs:** Helps reduce the risk of bugs, as multiple pairs of eyes can catch potential issues that a single developer may have missed
- **Improved code review process:** Improves the code review process, as multiple developers can review and provide feedback

By bringing various perspectives to the development process, reducing the risk of bugs and improving the code review process, collaborating with other developers can help ensure that you produce high-quality code that is easy to maintain.

Unit Testing Principles

Unit testing

- ➔ Unit testing, also known as component testing, is a critical quality measure in IT aimed at delivering quality software rapidly.
  - ◆ The main goal of unit testing is to verify that each code implementation performs its intended function.
- ➔ Involves testing the smallest units of code, called modules or components, individually.
- ➔ Allows developers to isolate each component to identify and fix issues early in the development lifecycle.

Best Practices:

- Aim to test each part of the software as early as possible.
- Create test cases that validate both valid and invalid inputs (e.g., testing password requirements).

Testing Example:

- For a password field requiring at least 8 characters, alphanumeric characters, and one symbol:
  - “Secret123!” should pass.
  - “secret01” should fail.

Quality of Tests:

- Well-written tests are valuable assets, while poorly written tests can become burdensome.
- Following unit testing principles helps in creating effective tests that are worth the investment.

Principles

The aim of DevOps teams is to build quality in. Testing the code on desired functionality and quality while writing the code is therefore an effective and efficient way to verify that the implementation actually does what it is intended to do.

Some benefits of unit testing are:

- It provides a fast feedback-loop for verifying code changes.
- It provides a safety net – we know that the code works.
- It reduces costs and technical debt (e.g. because it reduces rework).
- It can be applied as regression test.
- It forms the basis for test automation.
- It facilitates easy verification of changes during maintenance.

FIRST-U rules

The **FIRST** principles, often referred to as the **FIRST-U** rules, are guidelines to help ensure effective unit testing. The acronym stands for:

- ➢ **F**ast
- ➢ **I**solated/Independent
- ➢ **R**epeatable
- ➢ **S**elf-validating
- ➢ **T**imely
- ➢ **U**nderstandable

Fast:

- Unit tests should execute quickly to avoid slowing down development and deployment processes.
- In Test-Driven Development (TDD), quick iterations of code changes and tests are essential for effectiveness.
- If unit tests are slow, the benefits of TDD diminish.
- On large systems, running thousands of unit tests can add up; for example, 2000 tests taking 200 milliseconds each would require 6.5 minutes to complete.
- While 6.5 minutes may seem manageable, running tests multiple times daily accumulates significant time.
- As the number of tests grows with new features, execution time increases, reducing the tests' value in providing timely feedback on system health.

Isolated/Independent:

- Avoid writing tests that depend on other test cases to prevent false alarms.
- Dependencies can complicate debugging and waste time identifying the source of failures.
- Each test should be runnable at any time and in any order, enhancing test reliability.
- Independent tests focus on specific behaviors, making it easier to pinpoint issues when they fail.
- Apply the Single Responsibility Principle (SRP) to tests, ensuring each test case has a single purpose.
- If a test case can fail for multiple reasons, consider splitting it into separate test cases for clarity.

Repeatable:

- A repeatable test consistently produces the same result every time it runs.
- To achieve repeatability, isolate tests from external factors beyond your control by using mock objects.
- For tests requiring interaction with external elements (e.g., databases), set up a private sandbox to prevent conflicts with other tests.
- A good practice is to utilize in-memory databases for this purpose.

Self-validating:

- Tests must be self-validating, meaning they can automatically verify whether the actual output matches the expected output.
- Each test should independently determine pass or fail status without manual interpretation of results.
- Avoid designs requiring manual setup steps; automate all necessary preparations for the test.
- Use an in-memory database to create schemas and insert dummy data for testing, ensuring consistent execution and results unaffected by external factors.

Timely:

- Unit tests can be written at any point but are most effective when done in a timely manner.
- Following Test Driven Development (TDD) is recommended for better integration of testing and coding.
- Establish guidelines or strict rules for unit testing, potentially using review processes or automated tools to reject code lacking sufficient tests.
- Writing smaller chunks of code before corresponding unit tests enhances test ease and provides immediate feedback as code behavior develops.

Understandable:

- Each unit test should have a descriptive title or user story that explains its purpose and expected outcomes.
- Avoid using simple numeric names (e.g., test1, test2) for tests; instead, assign meaningful and useful names that convey the test's intent.

What Is Continuous Integration?

Continuous Integration (CI):

- A software development practice involving frequent integration of code changes into a shared repository, often multiple times a day.
- Aims to detect and resolve integration issues early, resulting in quicker development cycles, better collaboration, and higher software quality.

CI Workflow:

- Developers commit code changes to a shared repository, triggering automated build and testing processes.
- These processes verify that new code integrates well with the existing codebase without introducing bugs or conflicts.
- Immediate reporting of any detected issues allows for quick resolution by developers.

Relation to DevOps:

- CI is part of a broader series of guides on DevOps practices.

Pillars of Continuous Integration

1. **Source Control:**
- Tracks and manages changes to the codebase, allowing simultaneous work on different features or bug fixes by developers.
- Facilitates collaboration, provides a history of code changes, and enables easy reversion to previous versions.
- Maintains a unified codebase, essential for continuous integration.
- Key features include branching, merging, conflict resolution, and tagging.

2. **Automated Testing:**
- Essential for quickly identifying and resolving code issues within continuous integration.
- Automatically runs tests (unit, integration, functional, performance) when developers commit changes to validate functionality and integration.
- Reduces manual testing efforts, provides rapid feedback, and helps maintain high code quality.

3. Build Automation:

- Involves using tools and scripts to compile code, manage dependencies, and package software for distribution or deployment.
- Triggered by changes to the codebase to ensure software is always in a releasable state.
- Helps detect compilation errors, missing dependencies, and packaging issues early, increasing efficiency and reducing deployment risks.

4. Security Scanning:

- Automatically identifies vulnerabilities (weak passwords, insecure configurations, outdated dependencies) in the codebase.
- Integrated into the CI pipeline to allow quick addressing of security issues.
- Minimizes risks and ensures adherence to security best practices throughout the development process.

What Are CI Servers?

A continuous integration (CI) server is a dedicated system or service that automates the processes of integrating, building, testing, and deploying code changes in a software development project. The CI server monitors the code repository for any new commits or changes, and when changes are detected, it automatically triggers the predefined CI pipeline.

The primary functions of a CI server include:

- Fetching the latest code from the shared repository.
- Compiling the code and managing dependencies.
- Executing automated tests (unit, integration, functional, etc.) to validate the code.
- Reporting any issues or errors encountered during the build or test process.
- Packaging the software for deployment, if all tests and builds are successful.

By automating these tasks, a CI server ensures that the codebase remains in a releasable state, reduces the risk of integration issues, and provides rapid feedback on the success or failure of code changes. This allows developers to quickly address any problems, leading to a more efficient development process and a higher-quality software product.

Continuous Integration: Benefits and Challenges

Benefits of continuous integration:

- **Improved code quality:** With frequent integration and automated testing, CI helps maintain a higher standard of code quality, leading to a more reliable and stable software product.
- **Faster time-to-market:** CI accelerates the development process by providing quick feedback, enabling teams to iterate more rapidly and deliver new features and bug fixes to users sooner.
- **Enhanced collaboration:** CI fosters a collaborative development environment, with shared ownership of the codebase, leading to improved communication and teamwork across the project.
- **Cost reduction:** By identifying and addressing issues early in the development process, CI reduces the cost associated with fixing bugs and integration problems later in the project lifecycle.

Challenges of continuous integration:

- **Initial setup and configuration:** Setting up a CI server, configuring pipelines, and integrating various tools can be time-consuming and complex, especially for large projects or teams new to CI.
- **Test suite quality:** A successful CI process relies on a comprehensive and well-maintained test suite. Developing and maintaining a robust set of tests can be challenging and time-consuming.
- **Resistance to change:** Implementing CI may require significant changes to existing workflows and processes. This can lead to resistance from team members who are accustomed to traditional development practices.
- **Infrastructure requirements:** CI can demand substantial computing resources, particularly for large projects with complex build and test processes. This may require investment in additional hardware or cloud infrastructure.

Continuous Integration Tools You Should Know

- ➔ Codefresh
- ➔ Jenkins
- ➔ GitHub Actions
- ➔ Bitbucket
- ➔ CircleCI
- ➔ GitLab CI

5 Continuous Integration Best Practices

1. **Integrate Early and Often:**

- Developers should frequently commit code changes to a shared repository, ideally multiple times a day.
- This practice prevents integration conflicts, allows for rapid identification and resolution of issues, and keeps the codebase up-to-date and releasable.

2. **Keep the Build Green at All Times:**

- The codebase should always be in a stable and releasable state, indicated by a "green" build.
- Developers must fix broken builds or failed tests immediately to maintain this state.
- A green build fosters team accountability and ensures prompt issue resolution, leading to a more efficient development process.

3. Write Tests as Part of Your Stories:

- Developers should create and maintain automated tests alongside new feature or bug fix development.
- This practice ensures tests are relevant and cover the latest code changes, validating requirements and improving code quality while preventing regressions.

4. Use Code Coverage to Find Untested Code:

- Code coverage measures the percentage of code executed by automated tests.
- Monitoring this metric helps identify untested or under-tested areas, allowing teams to target those for additional testing.
- Prioritizing testing efforts based on code coverage enhances the overall quality of the test suite.

5. Scan for Security Issues and Vulnerabilities for Each Code Change:

- Incorporating security checks into the CI process helps identify potential weaknesses in the codebase.
- Performing security scans with each code change allows teams to catch vulnerabilities early, making them easier and less costly to fix.
- This practice fosters a security-conscious culture and prioritizes security throughout the development lifecycle.

Must Read:

https://medium.com/@melissavaiden/seed-data-for-your-database-f3f0cb868f1a

https://thefellowcoder.medium.com/10-best-tools-and-practices-for-boosting-code-quality-in-node-js-3c4949f991d1

https://medium.com/@kaanfurkanc/unit-testing-best-practices-3a8b0ddd88b5

https://www.perforce.com/blog/qac/what-is-linting#:~:text=Linting%20is%20the%20automated%20checking,a%20Unix%20utility%20for%20C