Enabling High-Quality Code in .NET

What is code quality?

When we talk about code quality, we can think of different aspects of it. We mainly refer to the attributes and characteristics of code. While there is no widely adopted definition of high-quality code, we know some of

Characteristics of good code:

- **It is clean.**
- **Absence of code smells.**
- **Consistent.**
- **Functional - it does what we say it does.**
- **Easy to understand.**
- **Secure.**
- **Efficient.**
- **Testable.**
- **Easy to maintain.**
- **Well documented.**

There are more characteristics of good code, but these are the core of **high-quality code**.

However, even though we know some attributes and characteristics of high-quality code, they can mean a different thing to different people or have a different view on priorities against them. However, we need to impose some standards and guidelines to promote a similar view of the topic of high-quality code. Automating them as much as possible is essential without leaving room for mistakes.
How can we enforce high code quality in our projects?

1.**Code reviews:** Implement regular code reviews to catch issues early and ensure adherence to coding standards.

2.**Static code analysis**: Use tools like SonarQube or Resharper to enforce coding standards and detect code smells.

3.Automated testing: Set up a comprehensive suite of automated tests, including unit, integration, and end-to-end tests.

4.Continuous Integration/Continuous Deployment (CI/CD): Use CI/CD pipelines (e.g., Azure DevOps, Jenkins) to build, test, and deploy code automatically.

5.Coding standards: Establish and enforce coding standards and best practices. This can be achieved through style guides, linting tools, and automated code formatters.

So, let's discuss some of these most critical practices.

Coding standards

Coding standards and code styles provide uniformity in code inside a project or a team. By using it, we can easily use and maintain our code. Style guides are company standard conventions, usually defined per programming language, as best practices that should be enforced. This ensures **consistent code across all team members.**

There are different examples of code standards, e.g., by a company - Microsoft, or per language, e.g., C#.

These standards **usually define the following**:

- Layout conventions.
- Commenting conventions.
- Language guidelines.
- Security.
- Etc.

ETC....like

- Indentation
- inline comments
- Use of global
- Structured programming
- Naming convension
- Error and exception handling
- proper logging implementation

Code reviews:

Code reviews are a helpful tool for teams to improve code quality. It enables us to reduce development costs by catching issues in the code earlier and as a communication and knowledge-sharing tool. Code review, in particular, means reviewing other programming codes for mistakes and other code metrics and ensuring that all requirements are implemented correctly.
Code review usually starts with a submitted Pull Request (PR) for code to be added to a codebase. Then, one of the team members needs to review the code.
It is essential to do code reviews to improve coding skills, but it also represents a great **knowledge-sharing tool** for a team. In a typical code review, we should check for:

- **Code readability.**
- **Coding style.**
- **Code extensibility.**
- **Feature correctness.**
- **Naming.**
- **Code duplication.**
- **Test coverage.**

And more.

Code reviews can be implemented in different ways, from single-person reviews to pair programming (which is much better). However, these methods are usually time-intensive, so code quality check tools can **automate that process**.

Tools to check for code quality

When we talk about tools for checking code quality, there are two options: **dynamic and static code analysis**.

- **Dynamic code analysis** involves analyzing applications during execution and analyzing code for reliability, quality, and security. This kind of analysis helps us find issues related to application integration with database servers and other external services. The primary objective is to find problems and vulnerabilities in software that can be debugged. Different tools for dynamic code analysis exist, such as Microsoft IntelliTest, Java Pathfinder, or KLEE for C/C++.
- Another kind of analysis is **static code analysis**. Here, we analyze source code to identify different kinds of flaws. However, it doesn't give us a holistic view of the application, so it is recommended that we use it together with dynamic code analysis. For Static Code Analysis, we can use tools like SonarQube, Coverity, Parasoft, etc.

This post will focus on **static code analysis**, which can be automated and produce results even without developer input.

Static code analysis

Static code analysis is a way to check application source code **even before the program is run**. It is usually done by analyzing code with a given set of rules or coding standards. It addresses code bugs, vulnerabilities, smells, test coverage, and other coding standards. It also includes common developer errors that are often found during PR reviews.

It is usually incorporated at any stage into the software development lifecycle after the "code development" phase and before running tests. CI/CD pipelines usually include static analysis reports as a quality gate, even before merging PRs to a master branch.

We can use static code analysis with CI/CD pipelines during development. This involves using a different set of tools to understand what rules are being broken quickly. This enables us to fix code earlier in the development lifecycle and avoid builds that fail later.
During static code analysis, we also check for some code metrics, such as **Halstead Complexity Measures**, **Cognitive Complexity**, or **Cyclomatic Complexity**, which counts the number of linearly independent paths within your source code. The hypothesis is that the higher the cyclomatic complexity, the higher the chance of errors. Modern use of cyclomatic complexity is to improve software testability.

Java code analysis

Java code analysis involves examining Java source code to identify potential issues such as bugs, vulnerabilities, code smells, and deviations from best practices. This analysis aims to improve code quality, maintainability, and security. Here are some common approaches and tools for Java code analysis:

1. **Static Analysis Tools**:
   - **FindBugs**: A static analysis tool that detects bugs in Java code.
   - **Checkstyle**: A development tool that ensures code adherence to coding standards and conventions.
   - **PMD**: A source code analyzer that finds common programming flaws like unused variables, empty catch blocks, and more.
2. **SonarQube/SonarCloud**: These platforms provide comprehensive code analysis capabilities, including static code analysis, code duplication detection, and code coverage metrics. They support multiple languages, including Java, and offer a wide range of rules and plugins to ensure code quality and security.
3. **IDE Built-in Tools**:
   - **Eclipse**: Offers built-in static analysis tools like the Eclipse Checkstyle Plugin and PMD Plugin.
   - **IntelliJ IDEA**: Provides inspections and code analysis tools that highlight potential issues in the code.
4. **Code Review**: Manual code review by peers or team members is an essential part of code analysis. It helps identify issues that automated tools might miss and promotes knowledge sharing and collaboration within the team.
5. **Automated Testing**:
   - **Unit Tests**: Writing comprehensive unit tests can uncover bugs and ensure that code behaves as expected.
   - **Integration Tests**: These tests verify interactions between different components/modules of the application, ensuring that they work together correctly.
   - **Code Coverage Tools**: Tools like JaCoCo measure the extent to which your code is tested by your test suite, helping identify areas that lack sufficient test coverage.
6. **Security Scanners**:
   - **OWASP Dependency-Check**: Identifies known vulnerabilities in project dependencies.

    - **OWASP Find Security Bugs**: A security-focused plugin for FindBugs that detects security vulnerabilities in Java code.

These tools and practices can be integrated into the development workflow to continuously improve code quality, maintainability, and security throughout the software development lifecycle.