5 Code Refactoring Best Practices

1. Collaborate with Testers

Engaging the quality assurance (QA) team during the refactoring process ensures that the changes made do not introduce new bugs or negatively impact the software's functionality. The QA team can help validate that the refactored code behaves as expected, maintaining the software's quality and reliability.

2. Automate the Process

Utilizing automated tools for code analysis, testing, and refactoring can help streamline the process and minimize the risk of introducing errors. Tools like static code analyzers, linters, and automated testing frameworks can help identify code smells, enforce coding standards, and ensure the refactored code meets the required quality criteria.

3. Refactor in Small Steps

Making small, incremental changes during the refactoring process reduces the likelihood of introducing bugs or breaking the software. By focusing on one improvement at a time, you can more easily identify and fix issues, making it easier to maintain a stable codebase throughout the refactoring process.

4. Troubleshoot and Debug Separately

Refactoring should be kept separate from bug fixing and troubleshooting. Mixing the two can lead to confusion and complicate the process of identifying the root causes of issues. Address known bugs before refactoring, and treat any new issues that arise during refactoring as separate tasks to maintain clarity and focus.

5. Prioritize Code Deduplication

One of the primary goals of refactoring is to eliminate duplicated code, as it can lead to inconsistencies, increase maintenance complexity, and make the codebase more error-prone. Focusing on identifying and extracting common functionality into reusable components, such as abstract classes, interfaces, or utility methods, can greatly improve the maintainability and readability of the code.