ONE PAGE SUMMARY

IT & Software › Design Patterns › Refactoring

Introduction to Refactoring

Refactoring: Improving code without changing its functionality.

DEFINITION

> Refactoring is the process of restructuring existing computer code without altering its external behavior. It aims to improve code readability, reduce complexity, and enhance maintainability.

KEY CONCEPTS

Code Smells

Indicators that suggest the need for refactoring, such as duplicate code or long methods.

Refactoring Techniques

Common methods include renaming variables, extracting methods, and simplifying conditional expressions.

Test-Driven Development

A practice where tests are written before code to ensure functionality remains intact during refactoring.

Continuous Refactoring

The practice of regularly refactoring code as part of the development process to maintain quality.

Version Control

Using tools like Git to track changes and revert to previous versions if refactoring introduces issues.

EXAMPLES

→ Renaming a variable for clarity.

→ Extracting a method from a long function.

→ Replacing a conditional with polymorphism.

MEMORY TIPS

★ Think of refactoring as 'cleaning up' your code.

★ Use the acronym 'CLEAN' to remember: Clear, Logical, Efficient, Adaptable, Neat.

★ Visualize refactoring as a 'makeover' for your code.

COMMON MISTAKES

✗ Not writing tests before refactoring.

✗ Overcomplicating the refactoring process.

✗ Failing to document changes made during refactoring.

QUICK RECAP

Refactoring is essential for maintaining high-quality code. It involves restructuring code to improve its readability and maintainability while ensuring functionality remains unchanged. Regular refactoring helps prevent technical debt.