

# Advanced Programming

## Refactoring – Improving Code Quality

**Instructor:** Ali Najimi

**Author:** Hossein Masihi

**Department of Computer Engineering**

**Sharif University of Technology**

**Fall 2025**



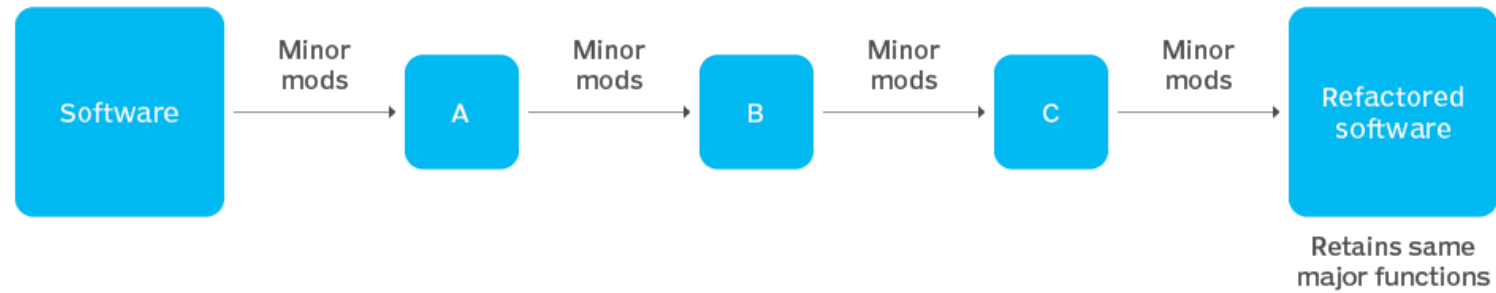


# Table of Contents

1. Refactoring — Concept
2. Clean Code Principles
3. Recognizing Bad Code Smells
4. Refactoring Techniques (Patterns)
5. Example
6. Summary



## The code refactoring process



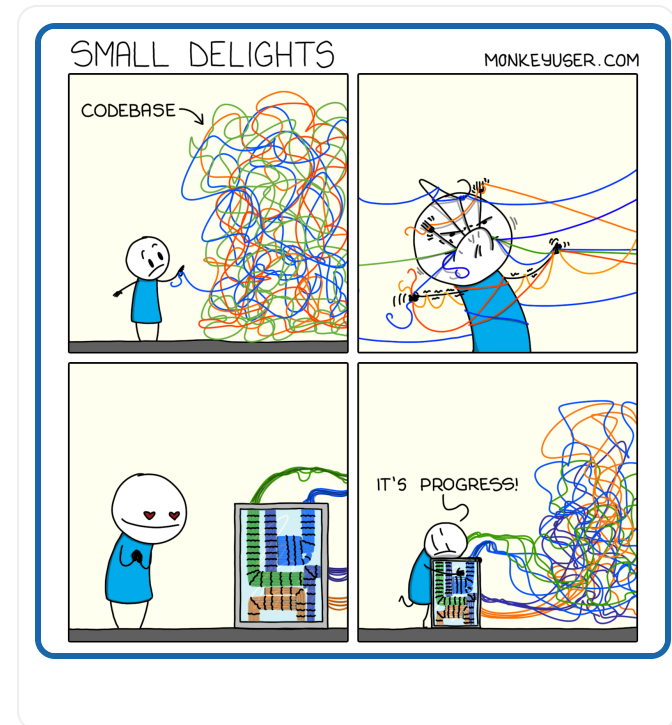
©2021 TECHTARGET. ALL RIGHTS RESERVED



# Refactoring — Concept

- **Refactoring** is the process of **improving internal code structure** *without changing external behavior.*
- Goal:
  - Cleaner structure
  - Better readability
  - Easier maintenance
  - Fewer bugs over time

"Refactoring is improving the design after the code is written." — Martin Fowler





# Clean Code Principles

Principle	Meaning
Readability	Code should be easy to understand
Single Responsibility	Each unit has one purpose
Small Methods	Methods should do <i>one thing</i>
Naming Matters	Clear, intention-revealing names
No Duplication	Reuse logic instead of copy-paste

Code is read **more often** than it is written.





## Example of Bad Code (Before Refactoring)

```
double calculateTotal(double price, double tax) {  
    double t = price * tax;  
    System.out.println("Total: " + (price + t));  
    return price + t;  
}
```

Problems:

- Mixed **calculation** and **printing**
- Ambiguous variable naming





# After Refactoring

```
double calculateTotal(double price, double tax) {  
    return price + taxAmount(price, tax);  
}  
  
double taxAmount(double price, double tax) {  
    return price * tax;  
}
```

- Clear naming
- Separated responsibilities
- Reusable logic

Cleaner code → easier testing and extension.



# Common Refactoring Patterns

Pattern	Purpose
<b>Extract Method</b>	Split large methods into smaller ones
<b>Rename Variable</b>	Improve meaning of identifiers
<b>Extract Class</b>	Move responsibilities into new class
<b>Inline Method</b>	Remove unnecessary methods
<b>Replace Magic Number with Constant</b>	Improve clarity and configurability



# Refactoring + Testing

- Refactor **only when tests exist**
- Unit tests guarantee behavior stays the same
- Refactoring should **not change output**

Testing + Refactoring = Safe Continuous Improvement



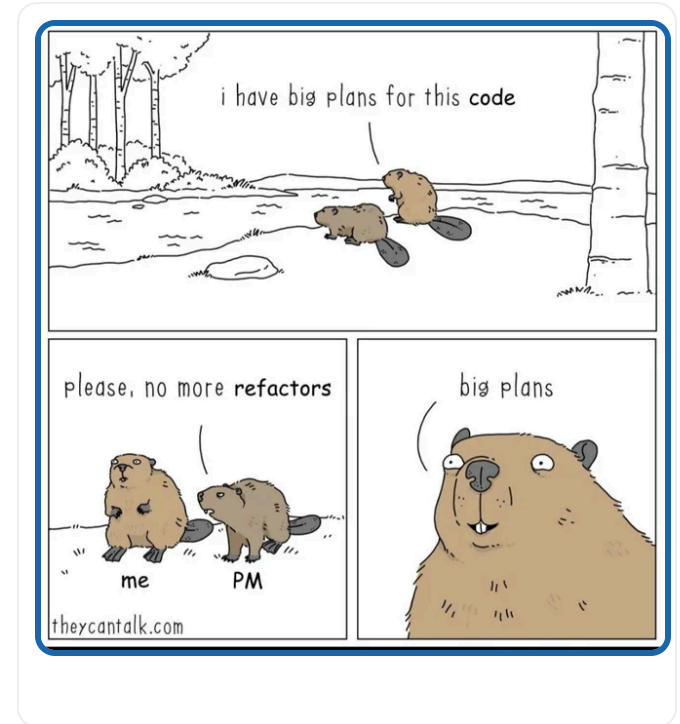
# Summary

Concept	Key Idea
Refactoring	Improve internal structure without changing behavior
Clean Code	Readable, simple, intention-revealing
Code Smells	Signals for needed improvement
Refactoring Patterns	Systematic ways to improve design

Great developers continually refactor — not just write code.

# Thank You!

Refactoring — Clean Code for the Future



Sharif University of Technology — Advanced Programming — Fall 2025