Advanced Topics & Large-Scale Strategy

Week 9: Discussion Session

Advanced Software Engineering

Learning Objectives

- Understand strategies for large-scale refactoring
- Learn to manage technical debt
- Plan and execute long-term improvements

Large-Scale Refactoring

Approaches

- Incremental vs. Big Bang
- Strangler Fig Pattern
- Parallel Change (Expand and Contract)

Managing Technical Debt

- Identifying and prioritizing debt
- Communicating debt to stakeholders
- Balancing business needs and code quality

Discussion Questions

1. What are the risks and benefits of incremental vs. big bang refactoring?
2. How do you prioritize technical debt in a large codebase? What frameworks or tools do you use?
3. How can you ensure business continuity during large-scale refactoring?
4. Share a real-world example of a successful (or failed) large-scale refactoring. What were the key lessons?

Code Examples

C++ Example: Strangler Fig Pattern

```cpp
// Legacy class
class LegacyOrderSystem {
public:
    void processOrder(const Order& order) {
        // ... legacy logic ...
    }
};

// New class
class NewOrderSystem {
public:
    void processOrder(const Order& order) {
        // ... improved logic ...
    }
};

// Facade
class OrderFacade {
    LegacyOrderSystem legacy;
    NewOrderSystem modern;
public:
    void processOrder(const Order& order, bool useNew) {
```

Key Takeaways

- Large-scale refactoring requires planning and communication
- Use patterns like Strangler Fig and Parallel Change
- Manage technical debt proactively

Required Reading

- Chapters 21-24 of "Working Effectively with Legacy Code"
- Focus on:
  - Large-scale refactoring strategies
  - Technical debt management
  - Communication and planning

Next Week

Prepare for hands-on kata:

- Applying large-scale refactoring
- Managing risk and business continuity
- Measuring success