Understanding Legacy Code

Week 1: Discussion Session

Advanced Software Engineering

Learning Objectives

- Define legacy code and its characteristics
- Understand the challenges and mindset needed
- Learn to approach legacy code systematically

What is Legacy Code?

"Code without tests" - Michael Feathers

Characteristics

- Difficult to change
- High coupling
- Low cohesion
- Insufficient documentation
- Missing or outdated tests

Discussion Questions

1. How does Feathers' definition of legacy code differ from the common industry perception? What implications does this have for how we treat our codebase?
2. In your experience, what are the most challenging aspects of working with legacy code? How do these challenges affect team productivity and morale?
3. What strategies have you used or seen used to deal with legacy code in your projects? How effective were they?

Code Examples

C++ Example: Tightly Coupled Legacy Code

```cpp
class DataProcessor {
    Database db;
    Logger logger;

public:
    void process(const std::string& data) {
        db.connect("hardcoded:connection:string");
        logger.log("Processing started");

        // Complex business logic intertwined with
        // database calls and logging
        if (data.length() > 0) {
            db.insert(data);
            logger.log("Data inserted");
        }

        db.disconnect();
    }
};
```

Python Example: Missing Abstraction

Key Takeaways

1. Legacy code is any code without tests, regardless of age
2. Working with legacy code requires a systematic approach
3. Understanding code dependencies is crucial
4. Small, safe changes are preferable to large refactoring

Required Reading

- Chapters 1-3 of “Working Effectively with Legacy Code”
- Focus on:
  - Understanding the legacy code dilemma
  - The role of testing
  - Basic strategies for working with legacy code

Next Week

Prepare for hands-on kata:

- Identifying legacy code characteristics
- Initial assessment techniques
- Setting up a testing strategy