Sprouting & Wrapping Kata

Week 6: Practical Session

Advanced Software Engineering

Learning Objectives

- Practice sprouting and wrapping techniques
- Make safe, incremental changes to legacy code
- Validate changes with tests

Kata Instructions

Work in pairs to apply sprouting and wrapping to the following code.

C++ Starting Point

```cpp
class UserManager {
public:
    bool authenticate(const std::string& user, const std::string& pass) {
        // Hardcoded credentials (legacy)
        return user == "admin" && pass == "password";
    }
};
```

Python Starting Point

```python
class UserManager:
    def authenticate(self, user, password):
        # Hardcoded credentials (legacy)
        return user == "admin" and password == "password"
```

Tasks

1. **Sprouting (20 min)**
   - Add new authentication logic in a separate method that uses a user database or config file
   - Make only the minimal change to the original method needed to call the new method
2. **Wrapping (20 min)**
   - Create a wrapper that logs authentication attempts
   - Ensure the wrapper can be tested independently
3. **Testing (30 min)**
   - Write tests for both the new method and the original authentication entry point
   - Validate that wrapping does not change legacy behavior
4. **Review (20 min)**
   - Share your approach with another pair
   - Discuss trade-offs and risks

Success Criteria

- New functionality added via sprouting
- Legacy code safely wrapped
- Comprehensive tests for both old and new code
- Risks and trade-offs discussed

Notes for Facilitators

- Encourage minimal changes to legacy code
- Focus on testability and safety
- Discuss real-world applications of these techniques

Follow-up

Prepare for next week's session on:

- Breaking dependencies
- Using adapters and seams
- Refactoring for flexibility