**ACS 560 Software Engineering**

**HW 11 Refactoring**

**Code Smell Addressed Summary**

The ItemDetailsForm and its test suite exhibited two primary code smells: **Tight Coupling** and **Lack of Separation of Concerns**. The test suite was heavily dependent on the internal implementation of the form, interacting directly with UI components like ComboBox, DatePicker, and Button. This led to fragile tests that could break even with minor UI changes, such as switching layout types or modifying component initialization logic. Additionally, the lack of abstraction resulted in repetitive and verbose code for tasks like setting field values, validating UI states, and handling events. This made the test suite harder to read, maintain, and extend as the form evolved.

Another issue was the **"Long Method"** smell in both the form and the test suite. Methods like testFormBindingAndValidation and handleSave were performing multiple responsibilities, combining UI setup, validation, and event handling logic into single, monolithic methods. This reduced readability and violated the Single Responsibility Principle, making it difficult to debug issues or extend functionality without introducing further complexity.

**Refactoring Approach**

1. **Loosened Coupling**:
   * The test suite was refactored to focus on observable outcomes rather than the internal implementation of the form. For instance, instead of verifying the exact UI structure or internal field bindings, the tests were updated to validate form behavior, such as the firing of AddEvent, UpdateEvent, and DeleteEvent.
   * The direct interaction with layout-specific components was minimized, ensuring that changes in the layout or component hierarchy (e.g., switching from HorizontalLayout to VerticalLayout) would not affect the test cases.
2. **Extracted Utility Methods**:
   * Common repetitive tasks, such as setting and validating field values, were extracted into helper methods. For example, setting ComboBox values and validating DatePicker behavior were encapsulated, reducing duplication and improving readability across test cases.
   * Event listeners and button click simulations were streamlined using higher-level abstractions, making the tests more concise and focused.
3. **Improved Separation of Concerns**:
   * The handleSave method in the form was split into distinct responsibilities for validation, event firing, and error handling. This reduced the complexity of the method and made it easier to test individual functionalities in isolation.
   * Similarly, test cases like testFormBindingAndValidation and testHandleSaveEventAdd were broken down to focus on individual behaviors, such as form binding, validation, or event handling.
4. **Centralized Validation Logic**:
   * Validation failure handling was standardized using consistent assertions. This ensured that errors like invalid itemPrice values were properly captured and displayed, while tests explicitly validated the form's behavior under these conditions.

**Impact of Refactoring**

The refactoring process brought several key benefits:

1. **Improved Test Maintainability**:
   * By decoupling test cases from specific layout and implementation details, the test suite now requires minimal updates when changes are made to the form's UI structure. For example, replacing HorizontalLayout with another layout type no longer impacts test functionality.
   * Repetitive logic was replaced with reusable utility methods, making the test suite easier to extend for new features or scenarios.
2. **Enhanced Readability and Clarity**:
   * Each test method now focuses on a single responsibility, such as validating a specific event or testing a single form behavior. This makes the tests easier to read, debug, and understand.
   * Refactoring long methods into smaller, focused methods improved the overall organization of both the form and its test suite.
3. **Increased Robustness**:
   * Decoupling test logic from implementation details ensures that tests validate functionality rather than incidental details, such as the internal binding of fields. This reduces the risk of false negatives caused by minor UI changes.
   * Centralized validation handling improved the reliability of error reporting, ensuring that failures are consistently logged and handled across the application.
4. **Alignment with Best Practices**:
   * By adhering to the principles of separation of concerns and single responsibility, the refactored code aligns with industry best practices, ensuring that the form and its test suite remain scalable and easy to maintain as the application grows.

In summary, this refactoring transformed the ItemDetailsForm and its test suite into a cleaner, more maintainable, and robust solution. The reduced coupling, improved modularity, and focus on observable outcomes ensure that the form's functionality is accurately tested while allowing for future changes with minimal impact on the test suite.

A screenshot of a computer

Description automatically generated