Modelarea partii experimentale

Overview of the Experimental Framework

The goal of this experimental study is to evaluate the efficiency, flexibility, and maintainability of the **Observer Design Pattern** when applied in a real-time notification system, compared to systems without the pattern. This framework tests the hypothesis that adopting the Observer Pattern reduces coupling, enhances scalability, and improves code maintainability.

---

1. Data Utilized in the Experiment

1.1 Entities Involved

- Subject: This is the object whose state will change (e.g., the price of a stock etc.). The subject's state changes, and all its observers must be notified of these changes.
- Observers: These are entities (objects, users, or systems) that need to be informed when the state of the subject changes. They act upon the changes in the subject's state (e.g., display updated stock prices).

1.2 Data Model

- Subject: Holds the state and a collection of observers. The state might be represented as a simple variable (e.g., stock price).
- Observers: Each observer is associated with a subject and is notified when the subject's state changes.
- Attributes:
    - Subject: The state attribute (e.g., price).
    - Observers: Listeners that react to changes in the subject's state (e.g., displaying the updated price).

1.3 Environment and Tools

- Programming Language: Python 3.x for implementation and testing.
- Testing Framework: unittest or pytest for validation of code functionality.
- Version Control: Git for code management.
- Containerization: Docker for a reproducible development environment (optional).

1.4 Data Set

- Small-Scale Data: Experiments are initially conducted with a small number of observers (e.g., 5 to 10).

- Large-Scale Data: Experiments with a larger number of observers (e.g., 100+). This helps to observe how the system scales and performs as the number of observers grows.

---

2. Experimental Procedure

2.1 System Setup

1. Without the Observer Pattern:
    - In this approach, the subject does not notify observers automatically. Observers need to poll the subject to check if there is a state change (manual querying).
2. With the Observer Pattern:
    - The subject maintains a list of observers. When the state changes, it automatically notifies each observer of the change (no polling).

2.2 Test Scenarios

To evaluate and compare the two approaches, we will perform the following experiments:

- Scenario 1: Maintainability
    - Add a new observer to the system and measure how much existing code must be modified.
    - Without Observer Pattern: Adding a new observer requires changes in the subject and observer classes.
    - With Observer Pattern: New observers can be added with minimal code changes, as the Observer Pattern decouples the observer from the subject.
- Scenario 2: Reusability
    - Reuse the subject and observer setup in a new context (e.g., different subject types like weather data).
    - Without Observer Pattern: Significant modifications may be required to adapt the system.
    - With Observer Pattern: Observers can be reused easily, and different types of subjects can be observed with minimal code changes.
- Scenario 3: Flexibility and Extensibility
    - Add or remove observers dynamically and measure the ease with which this can be done.
    - Without Observer Pattern: Modifying the list of observers may require significant changes to the subject's code.

    - With Observer Pattern: Observers can be added or removed dynamically without any changes to the core logic of the subject.

---

3. Validation of Results

3.1 Comparison to Literature

To validate the benefits of the Observer Pattern, we compare results with literature and existing studies. Specifically, we compare:

- Maintainability: How easy is it to modify or extend the codebase (adding/removing observers)?
- Scalability: How well does the system handle an increasing number of observers (does the system remain manageable and efficient)?
- Flexibility: How easy is it to introduce new observers or new types of subjects?

3.2 Metrics for Validation

To validate the two approaches, we will use the following metrics:

- Code Complexity: Measure the lines of code modified when adding/removing observers, or when extending the system with new subjects or observers.
- Execution Time: Measure the time it takes for the subject to notify all observers, for both small-scale and large-scale experiments.
- Developer Time: Measure how long it takes for a developer to add or remove observers in both approaches.

---

4. Mathematical Model of the Experiment

The time complexity of both approaches in terms of updating observers is O(N), where N is the number of observers:

Without the Observer Pattern:

- Each update requires each observer to poll the subject for updates. This results in O(N) time complexity for each state change.

With the Observer Pattern:

- The subject notifies each observer about the state change, which also results in O(N) time complexity. Each observer is notified exactly once.

Thus, in terms of time complexity, both approaches are O(N) for notifying observers. However, the real value of the Observer Pattern comes from:

1. Decoupling the subject from observers (easier to maintain).
2. Dynamic observer management (easier to add/remove observers).

3. Cleaner code due to the separation of concerns.

Final Report Structure (Proposed)

1. **Introduction to Software Design Patterns**
2. **Observer Pattern Overview**
3. **Experimental Model**
    - Data Utilized
    - Experimental Procedure
    - Validation of Results
    - Mathematical Model
    - Experiment Code
4. **Results and Discussion**
5. **Conclusion**
6. **References**