Modelarea partii experimentale

**1. 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.

**2. Data Description**

**2.1. Simulated Dataset**

The dataset simulates a real-world system where multiple entities (observers) subscribe to state changes in a central subject. For example:

* **Subject**: Represents entities like stocks in a market or users in a chatroom.
* **Observers**: Represent clients receiving updates, such as apps tracking stock prices or chatroom participants.

The dataset contains:

* **Subjects**:
  + A set of stocks: {"AAPL", "TSLA", "GOOGL"}.
  + Each stock has an initial price and periodic updates to its price.
* **Observers**:
  + Clients subscribed to updates. Each observer tracks one or more subjects.

**2.2. Synthetic Scenarios**

* Small-scale example: 3 subjects, 5 observers.
* Medium-scale example: 10 subjects, 50 observers.
* Large-scale example: 100 subjects, 1000 observers.

The data is designed to represent a growing system where new observers or subjects can be dynamically added.

**3. Experimental Design**

**3.1. Implementation Variants**

Two implementation strategies are evaluated:

1. **Without the Observer Design Pattern**:
   * A tightly coupled system where each observer explicitly queries the subject for updates.
   * Hardcoded relationships between subjects and observers.
2. **With the Observer Design Pattern**:
   * A decoupled system where subjects notify observers of changes automatically.
   * Dynamic subscription and removal of observers.

**3.2. Test Scenarios**

1. **Dynamic Subscription**:
   * Add new observers during runtime and measure the ease of integration.
2. **Notification Handling**:
   * Measure the time taken to notify observers of a change.
3. **Extensibility**:
   * Introduce a new type of observer and evaluate code changes required.

**4. Validation Methodology**

**4.1. Metrics for Evaluation**

1. **Coupling**: Use dependency analysis tools to compare the dependency graph of the system with and without the Observer Pattern.
2. **Scalability**: Measure notification time as the number of observers grows.
3. **Maintainability**: Assess the lines of code modified when adding a new observer or subject.

**4.2. Validation Process**

* Implement both variants of the system using Python.
* Use a profiler to measure execution time for notifications.
* Simulate adding/removing observers and evaluate the required code changes.

**4.3. Comparative Analysis**

* Compare results with studies in the literature that report on the coupling and scalability of real-time systems.

**5. Mathematical Model**

The experimental behavior is modeled as follows:

1. **Notification Time Without Observer Pattern**:
   * Each observer polls the subject for updates:  
     Twithout=O(N^2) where N is the number of observers.
2. **Notification Time With Observer Pattern**:
   * The subject actively pushes updates to observers:  
     Twith=O(N), demonstrating linear scalability.
3. **Coupling Metric**:
   * Coupling is measured by the number of dependencies:
     + Without Observer: Cwithout ∝ O(N×M), where M is the number of subjects.
     + With Observer: Cwith ∝ O(N+M)

**6. Expected Outcomes**

1. **Improved Scalability**:
   * Notification time increases linearly with observers in the Observer Pattern implementation.
2. **Reduced Coupling**:
   * The dependency graph of the Observer Pattern implementation is significantly simpler.
3. **Enhanced Maintainability**:
   * Code changes required for new observers or subjects are minimal.

**7. Subchapters**

This chapter can be divided into the following subchapters for clarity:

1. Data Description
2. Experimental Design
3. Validation Methodology
4. Mathematical Model
5. Expected Outcomes

**Analysis of Results**

1. **Scalability**:
   * Without the Observer Pattern, the time complexity grows quadratically O(N^2) as each observer queries the subject independently.
   * With the Observer Pattern, time complexity grows linearly O(N), as the subject efficiently notifies all observers.
2. **Maintainability**:
   * Without the Observer Pattern, adding or removing observers requires modifying existing code and increases coupling.
   * With the Observer Pattern, the decoupling allows dynamic subscription/removal with minimal changes.