Department of Computer Engineering

Academic Term: First Term 2023-24

Class: T.E /Computer Sem – V / Software Engineering

Practical No:	7
Title:	**Design Using Object-Oriented Approach with Emphasis on Cohesion and Coupling in Software Engineering**
Date of Performance:	12-09-23
Roll No:	
Team Members:	Ishita, Kashmira Aaron

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Total Score
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Theory Understanding(02)	02(Correct)	NA	01 (Tried)	
3	Content Quality (03)	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Questions (04)	04(done well)	3 (Partially Correct)	2(submitted)	

Signature of the Teacher:

Department of Computer Engineering

Academic Term: First Term 2022-23

Class: T.E /Computer Sem – V / Software Engineering

Signature of the Teacher:

Lab Experiment 07

Experiment Name: Design Using Object-Oriented Approach with Emphasis on Cohesion and Coupling in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the Object-Oriented (OO) approach in software design, focusing on the principles of cohesion and coupling. Students will gain practical experience in designing a sample software project using OO principles to achieve high cohesion and low coupling, promoting maintainable and flexible software.

Introduction: The Object-Oriented approach is a powerful paradigm in software design, emphasizing the organization of code into objects, classes, and interactions. Cohesion and Coupling are essential design principles that guide the creation of well-structured and modular software.

Lab Experiment Overview:

1. Introduction to Object-Oriented Design: The lab session begins with an introduction to the Object-Oriented approach, explaining the concepts of classes, objects, inheritance, polymorphism, and encapsulation.
2. Defining the Sample Project: Students are provided with a sample software project that requires design and implementation. The project may involve multiple modules or functionalities.
3. Cohesion in Design: Students learn about Cohesion, the degree to which elements within a module or class belong together. They understand the different types of cohesion, such as functional, sequential, communicational, and temporal, and how to achieve high cohesion in their design.
4. Coupling in Design: Students explore Coupling, the degree of interdependence between modules or classes. They understand the types of coupling, such as content, common, control, and stamp coupling, and strive for low coupling in their design.
5. Applying OO Principles: Using the Object-Oriented approach, students design classes and identify their attributes, methods, and interactions. They ensure that classes have high cohesion and are loosely coupled.
6. Class Diagrams: Students create Class Diagrams to visually represent their design, illustrating the relationships between classes and their attributes and methods.
7. Design Review: Students conduct a design review session, where they present their Class Diagrams and receive feedback from their peers.
8. Conclusion and Reflection: Students discuss the significance of Object-Oriented Design principles, Cohesion, and Coupling in creating maintainable and flexible software. They reflect on their experience in applying these principles during the design process.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the Object-Oriented approach and its core principles, such as encapsulation, inheritance, and polymorphism.
- Gain practical experience in designing software using OO principles with an emphasis on Cohesion and Coupling.

- Learn to identify and implement high cohesion and low coupling in their design, promoting modular and maintainable code.
- Develop skills in creating Class Diagrams to visualize the relationships between classes.
- Appreciate the importance of design principles in creating robust and adaptable software.

Pre-Lab Preparations: Before the lab session, students should review Object-Oriented concepts, such as classes, objects, inheritance, and polymorphism. They should also familiarize themselves with the principles of Cohesion and Coupling in software design.

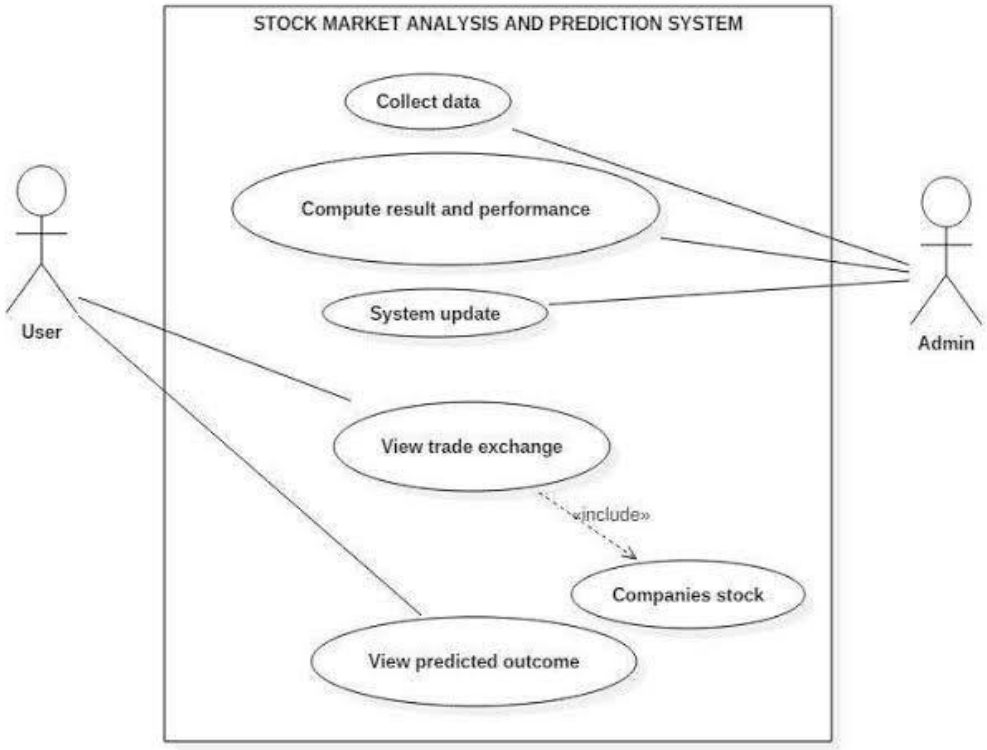
Materials and Resources:

- Project brief and details for the sample software project
- Whiteboard or projector for creating Class Diagrams
- Drawing tools or software for visualizing the design

Conclusion: The lab experiment on designing software using the Object-Oriented approach with a focus on Cohesion and Coupling provides students with essential skills in creating well-structured and maintainable software. By applying OO principles and ensuring high cohesion and low coupling, students design flexible and reusable code, facilitating future changes and enhancements. The experience in creating Class Diagrams enhances their ability to visualize and communicate their design effectively. The lab experiment encourages students to adopt design best practices, promoting modular and efficient software development in their future projects. Emphasizing Cohesion and Coupling in the Object-Oriented approach empowers students to create high-quality software that meets user requirements and adapts to evolving needs with ease.

TE COMPS B

SE EXP 7: Architecture



SHORTCOMINGS For Stock Market Prediction model:

1. Data Quality: Models can be sensitive to low-quality or noisy data, leading to inaccurate predictions.
2. Market Volatility: ML models may struggle to predict extreme market events or sudden changes in market sentiment.
3. Non-Stationarity: Stock market data is non-stationary, making it challenging for traditional models to adapt.
4. Overfitting: ML models can overfit to historical data, leading to poor generalization.
5. Feature Engineering: Selecting relevant features is crucial and can be time-consuming.
6. Market Noise: Noise in financial markets can lead to model inaccuracies.
7. Regulatory Changes: Changes in regulations and market dynamics can impact model performance.
8. Limited Data: Historical data is limited and may not capture all relevant market conditions.
9. Model Interpretability: Many ML models lack transparency, making it difficult to understand their predictions.
10. Evaluation Metrics: Choosing appropriate evaluation metrics can be challenging due to market dynamics.

UPDATES For Stock Market Prediction model:

1. Data Quality: Improve data preprocessing techniques, incorporate data from multiple sources, and use data cleansing methods.
2. Market Volatility: Incorporate advanced modeling techniques, such as deep learning and recurrent neural networks.
3. Non-Stationarity: Use time-series analysis techniques and consider online learning approaches.
4. Overfitting: Regularize models, use larger and more diverse datasets, and implement cross-validation.
5. Feature Engineering: Utilize automated feature selection methods and explore deep learning models.

6. Market Noise: Apply filtering techniques and consider sentiment analysis of news and social media data.
7. Regulatory Changes: Monitor regulatory changes and incorporate macroeconomic indicators.
8. Limited Data: Explore alternative data sources and consider synthetic data generation.
9. Model Interpretability: Develop interpretable models or employ interpretability techniques.
10. Evaluation Metrics: Explore alternative evaluation metrics like Sharpe ratio or Sortino ratio.
11. Ensemble Models: Consider using ensemble methods like bagging and boosting to improve accuracy and robustness.

POSTLABS:

a) Analyse a given software design and assess the level of cohesion and coupling, identifying potential areas for improvement:

The software design of the "Samachaar" website demonstrates reasonably good levels of cohesion and coupling. Cohesion is apparent in well-defined components with distinct functions, while loose coupling allows for flexibility and minimal interdependence between components. To improve the design, the website can benefit from further enhancing cohesion by minimizing overlapping functions and ensuring each component has a single, clear responsibility. Additionally, refining interfaces and interactions between components, focusing on modularity, and conducting regular testing and refactoring efforts are recommended for ongoing software quality and maintainability.

b) Apply Object-Oriented principles, such as encapsulation and inheritance, to design a class hierarchy for a specific problem domain.
In the "Vehicle" domain, we establish a class hierarchy using Object-Oriented principles:

1. Vehicle (Base Class):
   - Properties: make, model, year
   - Methods: start, stop, accelerate, brake

2. Car (Inherits from Vehicle):
   - Additional Properties: numDoors, fuelType
   - Additional Methods: lockDoors, unlockDoors

3. Motorcycle (Inherits from Vehicle):
   - Additional Properties: hasHelmetStorage
   - Additional Methods: putOnHelmet, takeOffHelmet

4. Truck (Inherits from Vehicle):
   - Additional Properties: cargoCapacity
   - Additional Methods: loadCargo, unloadCargo

This hierarchy exemplifies encapsulation, where properties and methods are contained within each class, and inheritance, which allows specialized classes to inherit properties and methods from the base class, promoting code reusability and structure.

c) Evaluate the impact of cohesion and coupling on software maintenance, extensibility, and reusability in a real-world project scenario.
In a real-world project, cohesion and coupling have significant effects:

- Software Maintenance: High cohesion simplifies changes, and low coupling reduces unintended impacts during maintenance.

- Software Extensibility: High cohesion and low coupling ease the addition of new features and components.

- Software Reusability: Well-structured, cohesive, and loosely coupled code is more reusable in various contexts.

In practice, striking a balance between cohesion and coupling is crucial for business agility, cost savings**,** team collaboration, and quality assurance in long-term projects.