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
Date of Performance:	
Roll No:	9563
Team Members:	Sanika Patankar, Lisa Gonsalves, Eden Evelyn Charles

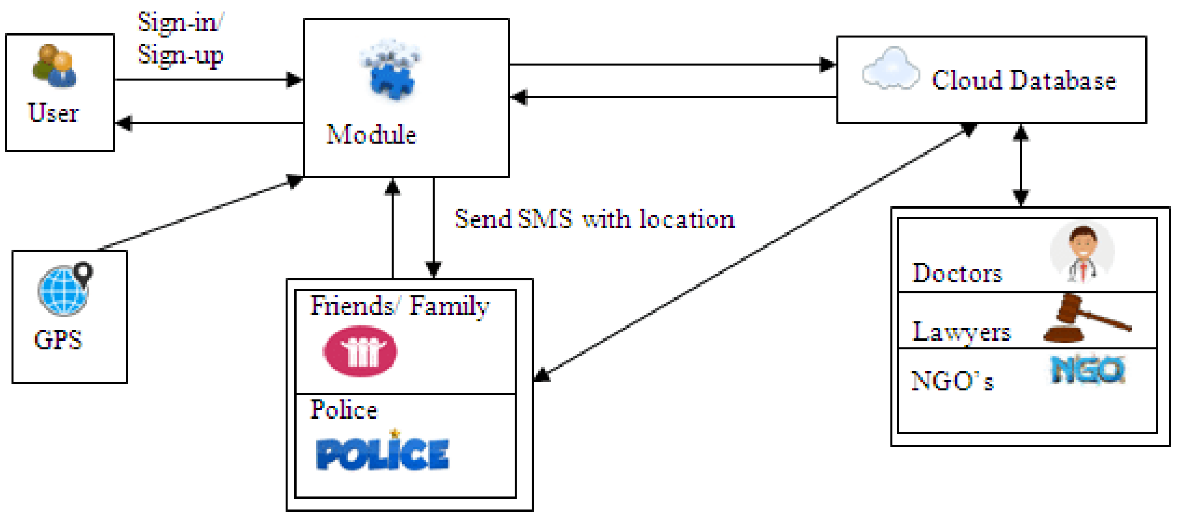
Rubrics for Evaluation:

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)	01 (All used)	02 (Partial)	03 (Rarely allowed)	
4	Post Lab Questions (04)	04 (Done Well)	03 (Partially Correct)	02 (Submitted)	

Signature of the Teacher:

Sanika Patankar
9563
TE COMPS B

SE EXP 7: Design using Object Oriented approach with emphasis on Cohesion and Coupling



SHORTCOMINGS For Women Safety and Period Management App:

1. **Inaccurate Predictions**: Period tracking apps often use algorithms and user input to predict menstrual and ovulation dates, which can be unreliable for those with irregular cycles.
2. **Lack of Inclusivity**: Many apps are designed for cisgender, heterosexual users, excluding LGBTQ+ individuals and those with non-binary gender identities.
3. **Privacy Concerns**: Some apps collect and share user data without consent, posing privacy and security risks, especially for sensitive information.
4. **Limited Educational Resources**: Many apps lack comprehensive information on reproductive health, sexual education, and healthy relationships.
5. **Cultural Sensitivity**: Apps may not consider cultural and regional variations in menstrual practices, potentially leading to insensitivity or misinformation.
6. **Insufficient Safety Features**: Some women's safety apps lack critical features like real-time location tracking and self-defense resources.
7. **Limited Accessibility**: Accessibility features for different languages, disabilities, and user-friendly design may be inadequate.
8. **Reliance on Technology**: The assumption of smartphone and internet access excludes those without these resources.
9. **Inadequate Mental Health Support**: Some apps focus on physical health but neglect mental health aspects, like addressing PMS and emotional impacts.
10. **Sustainability Concerns**: Some apps promote disposable products without considering eco-friendly options.
11. **User Engagement and Retention**: Many apps struggle to maintain long-term user engagement, limiting their benefits.

12. **Stereotyping and Stigmatization**: Some apps perpetuate stereotypes and stigma surrounding menstruation and women's safety, contributing to negative cultural perspectives.

UPDATES For Women Safety and Period Management App:

1. **Enhanced Privacy and Security:**
   - Implement robust data encryption and user consent mechanisms.
   - Enable users to manage data sharing with third parties.
   - Educate users about privacy settings and data protection.
2. **Inclusivity and Diversity:**
   - Foster inclusivity for LGBTQ+ individuals and those with diverse gender identities.
   - Offer content in multiple languages for a global audience.
   - Provide information on different cultural practices related to menstruation.
3. **Accuracy and Customization:**
   - Improve period prediction accuracy with AI and machine learning.
   - Allow users to input additional health data for more precise predictions.
   - Offer customizable settings for varying cycle lengths and irregularities.
4. **Educational Resources:**
   - Provide comprehensive and evidence-based information on menstrual health and safety.
   - Include educational content on reproductive health, sexual education, and healthy relationships.
   - Keep users informed and engaged with articles, videos, and quizzes.
5. **Safety Features:**
   - Incorporate real-time location tracking and safety alerts.
   - Enable quick access to emergency contacts and authorities.
   - Provide self-defence resources and safety tips.
6. **Accessibility:**
   - Ensure accessibility for people with disabilities through screen readers and voice commands.
   - Offer multilingual support and culturally sensitive design.
7. **Mental Health Support:**
   - Integrate features for tracking and managing PMS symptoms and emotional changes.
   - Include mindfulness exercises, stress-reduction techniques, and resources for addressing mental health challenges related to menstruation and safety.
8. **Sustainability:**
   - Promote eco-friendly period products and sustainable practices.
   - Educate users on the environmental impact of different menstrual products.
9. **User Engagement and Retention:**
   - Create a community feature for users to connect, share experiences, and seek support.
   - Send personalised reminders and notifications based on users' preferences and cycle tracking.
10. **Mood and Symptom Tracking:**

    - Allow users to log and track emotional changes and physical symptoms throughout their menstrual cycle.
    - Provide insights and recommendations based on users' symptom patterns.
11. **User Feedback and Continuous Improvement:**
    - Encourage user feedback and suggestions for ongoing app enhancements.
    - Regularly update the app to meet evolving user needs.
12. **Integration with Healthcare Providers:**
    - Enable users to share data with healthcare professionals for better health management and consultations.

POSTLABS:

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

Cohesion and coupling analysis in software design involves understanding how components relate to each other. Cohesion measures how closely related elements are within a module, while coupling evaluates the interdependence between different modules. To assess and improve these aspects:

Assessing Cohesion:

1. **Functional Cohesion (High)**: Ensure components have a single, well-defined purpose to avoid mixing unrelated functionality.
2. **Sequential Cohesion (Low)**: Separate functions in a module with specific execution sequences to enhance modularity.
3. **Communicational Cohesion (Medium)**: Organize functions dealing with related data into separate components.
4. **Procedural Cohesion (Medium)**: Arrange functions based on related functionality rather than execution order.

Assessing Coupling:

1. **Low Coupling (Good)**: Minimize interdependence between modules with well-defined interfaces.
2. **Content Coupling (High)**: Hide internal data of components using encapsulation and clear interfaces.
3. **Common Coupling (Medium)**: Handle shared data safely to avoid data conflicts.
4. **Control Coupling (Medium)**: Use structured interfaces to reduce one component's influence on another.

Improvement Strategies:

1. **Refactoring**: Split low-cohesion components into smaller, focused ones.

2. **Encapsulation**: Hide data to reduce content coupling and expose only necessary interfaces.
3. **Clear Interfaces**: Ensure components communicate through well-defined, documented interfaces.
4. **Modularization**: Divide the software into independent, manageable modules.
5. **Dependency Injection**: Use dependency injection or inversion of control for decoupling.
6. **Design Patterns**: Implement patterns like Singleton and Observer to control instantiation and communication more decoupled.

Specific improvements depend on the software's context, with the goal of achieving a balance between cohesion and coupling for a modular and maintainable design.

b. Apply Object-Oriented principles, such as encapsulation and inheritance, to design a class hierarchy for a specific problem domain.

Design a simple class hierarchy for a problem domain related to "Vehicles." In this hierarchy, we'll use Object-Oriented principles like encapsulation and inheritance to create a structured representation of various types of vehicles.

Class Hierarchy:

1. Vehicle (Base Class):
   - This is the top-level class representing all vehicles. It includes common attributes and methods that are applicable to all vehicles.

~~~~
  ```python
  class Vehicle:
      def __init__(self, make, model, year):
          self.make = make
          self.model = model
          self.year = year

      def start(self):
          pass  # Method to start the vehicle

      def stop(self):
          pass  # Method to stop the vehicle
  ```
~~~~

2. Car (Inherits from Vehicle):
   - This class represents specific types of vehicles, like cars. It inherits common attributes and methods from the `Vehicle` class but may have additional attributes and methods unique to cars.

~~~~
  ```python
  class Car(Vehicle):
      def __init__(self, make, model, year, num_doors):
          super().__init__(make, model, year)
          self.num_doors = num_doors

      def honk_horn(self):
          pass  # Method to honk the car's horn
  ```
~~~~

3. Motorcycle (Inherits from Vehicle):
   - Similar to the `Car` class, the `Motorcycle` class represents a specific type of vehicle. It inherits attributes and methods from the `Vehicle` class but may have unique characteristics for motorcycles.

```python
class Motorcycle(Vehicle):
    def __init__(self, make, model, year, engine_size):
        super().__init__(make, model, year)
        self.engine_size = engine_size

    def wheelie(self):
        pass  # Method to perform a wheelie (example method for motorcycles)
```

In this class hierarchy:
- The `Vehicle` class is the base class, encapsulating common attributes (make, model, year) and methods (start, stop).
- The `Car` and `Motorcycle` classes inherit from `Vehicle`, inheriting these common attributes and methods while adding their specific attributes and methods.
- Encapsulation is achieved by making attributes private (by convention, by prefixing them with an underscore) and providing methods to access and manipulate the attributes. For example, you could add getter and setter methods for attributes as needed.

This class hierarchy demonstrates encapsulation and inheritance, two fundamental principles of Object-Oriented Programming (OOP). It provides a structured way to represent vehicles in a program, making it easy to create instances of different types of vehicles and call their respective methods.

c. Evaluate the impact of cohesion and coupling on software maintenance, extensibility, and reusability in a real-world project scenario.

Cohesion and coupling have significant effects on software maintenance, extensibility, and reusability in real-world projects.

Cohesion:

- **Impact on Maintenance:** High cohesion makes maintenance easier by grouping related functionality, while low cohesion scatters unrelated functions, making maintenance more challenging.
- **Impact on Extensibility:** High cohesion simplifies adding new features, while low cohesion hinders extensibility by requiring code reorganization.
- **Impact on Reusability:** High cohesion enhances reusability, while low cohesion reduces it as extracting parts of a module becomes difficult.

Coupling:

- **Impact on Maintenance:** Low coupling eases maintenance by reducing the risk of unintended consequences, while high coupling complicates it by requiring changes in multiple modules.

- **Impact on Extensibility:** Low coupling simplifies extensibility, allowing new features with minimal impact, while high coupling impedes it by necessitating extensive changes.
- **Impact on Reusability:** Low coupling enhances reusability, enabling the independent use of modules, while high coupling reduces reusability due to interdependencies.

Real-World Scenario: In a legacy e-commerce platform:

- **Cohesion:** Low cohesion scatters shopping cart logic, making maintenance and feature additions like multiple shipping addresses difficult. High cohesion, achieved through refactoring, consolidates shopping cart functionality, simplifying maintenance and feature additions.
- **Coupling:** High coupling between user authentication and product recommendation modules in the legacy code makes changing authentication impact recommendations. In the refactored version, low coupling separates these modules, allowing flexible changes to authentication without affecting recommendations.