UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE - 6th Sem)

Subject Name: System Design

Subject Code: 23CSH-314

SUBMITTED BY:
Name : Nagesh Kumar
UID: 23BCS10608

SUBMITTED TO:
Er.Alok Sir(E15012)
Section: KRG-1A

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+ Accredited University

ASSIGNMENT- 1

Student Name: Nagesh Kumar UID: 23BCS10608
Branch: BE-CSE Section/Group: KRG 1-A
Semester: 6th Date of Performance: 01-02-2026
Subject Name: System Design Subject Code: 23CSH-314

Q1. Explain SRP and OCP in detail with proper examples.
Ans :-

---

1. Single Responsibility Principle (SRP)

Definition:

The **Single Responsibility Principle (SRP)** states that **a class should have only one reason to change**.

In simpler terms, **a class should perform only one responsibility or one job**.

Each class or module should focus on **one business logic or concern** only.

Why SRP is Important?

1. **Easier to understand** – Code is simpler and more readable
2. **Easier to test** – Unit testing becomes straightforward
3. **Easier to maintain** – Changes are localized
4. **Fewer bugs** – Less chance of unintended side effects
5. **Better reusability** – Independent responsibilities can be reused

---

SRP Violation Scenario

When **one class handles multiple responsibilities**, it violates SRP.

One class doing multiple tasks:

- Data handling
- Report generation
- Email sending

Example (SRP Violation)

```
class Student {
public:
    void addStudent() {
        // Add student to database
    }

    void generateReport() {
        // Generate PDF report
    }

    void sendEmail() {
        // Send email to student
    }
};
```

Problems with this design:

- Database change affects email logic
- Report format change forces modification in Student class
- Difficult to test and maintain

SRP – Correct Design

```
class StudentService {
```

```
public:
    void addStudent();
};

class ReportService {
public:
    void generateReport();
};

class EmailService {
public:
    void sendEmail();
};
```

Advantages:

1. Each class has **one responsibility**
2. Changes in email logic won't affect reports
3. Better maintainability and scalability

---

2. Open/Closed Principle (OCP)

Definition:

The **Open/Closed Principle (OCP)** states that:

Software entities should be open for extension but closed for modification

This means we should be able to **add new functionality without changing existing tested code**.

Why OCP is Important?

- Prevents breaking existing logic
- Encourages scalable and extensible systems
- Supports plug-and-play architecture
- Reduces regression bugs

Example (OCP Violation)

```
class Payment {
public:
    void pay(string type) {
        if(type == "UPI") {
            // UPI payment logic
        }
        else if(type == "CARD") {
            // Card payment logic
        }
    }
};
```

Problems:

- Adding a new payment method requires modifying this class
- Multiple if-else conditions
- High risk of breaking existing functionality

OCP – Correct Design (Using Polymorphism)

```
class Payment {
public:
    virtual void pay() = 0;
};
```

```
class UpiPayment : public Payment {
public:
    void pay() override {
        // UPI logic
    }
};

class CardPayment : public Payment {
public:
    void pay() override {
        // Card logic
    }
};
```

Benefits:

- New payment methods can be added without modifying existing code
- Follows polymorphism
- Highly scalable and extensible

Q2. Discuss in detail about the violations in SRP and OCP along with their fixes.

Ans.

Software design principles are introduced to improve the **quality, maintainability, scalability, and reliability** of software systems. Among them, **Single Responsibility Principle (SRP)** and **Open/Closed Principle (OCP)** are frequently violated in real-world systems.

Below is a detailed discussion of their violations, causes, consequences, and fixes.

Part A: Violations of Single Responsibility Principle (SRP)

1. Meaning of SRP Violation

An SRP violation occurs when **a single class handles multiple responsibilities or business concerns**, giving it **multiple reasons to change**.

Common Causes of SRP Violation

1. **Lack of proper design planning**
2. **Procedural thinking in OOP systems**
3. **Time constraints and quick fixes**
4. **Overloaded controller/manager classes (God Classes)**

Characteristics / Symptoms of SRP Violation

- Large classes with too many methods
- Unrelated functionalities in one class
- Difficult to understand and test
- High coupling and low cohesion

Such classes are often called:

God Classes / Blob Classes

Consequences of SRP Violation

1. Poor maintainability
2. Low reusability
3. High risk of bugs
4. Difficult unit testing

5. Tight coupling between components

Fix for SRP Violations

Solution: Separation of Concerns

Steps:

- Identify distinct responsibilities
- Move each responsibility to a separate class

Techniques Used:

- Layered Architecture
- Service classes
- Repository pattern
- MVC (Model-View-Controller)

Part B: Violations of Open/Closed Principle (OCP)

1. Meaning of OCP Violation

An OCP violation occurs when **existing source code must be modified to add new functionality**, increasing the risk of regression.

Common Causes of OCP Violation

1. Excessive if-else or switch statements
2. Lack of abstraction
3. Poor use of polymorphism
4. Hard-coded business rules

Characteristics / Symptoms of OCP Violation

- Long conditional chains
- Frequent modification of the same file

- Ripple effect of changes
- Absence of interfaces or abstract classes

Consequences of OCP Violation

1. High regression risk
2. Poor scalability
3. Increased testing cost
4. Code fragility
5. Low reusability

Fix for OCP Violations
Solution: Abstraction and Polymorphism
Key Techniques:

- Interfaces and abstract classes
- Strategy pattern
- Factory pattern
- Dependency Inversion Principle
- Plugin-based architecture

Comparative Analysis (SRP vs OCP Violations)

Aspect	SRP Violation	OCP Violation
Core problem	Too many responsibilities	Not extensible
Main cause	Poor separation of concerns	Lack of abstraction

Aspect	SRP Violation	OCP Violation
Impact	Poor maintainability	Poor scalability
Main symptom	God classes	if-else chains
Primary fix	Split classes	Use interfaces
Design level	Structural issue	Behavioral issue

Q3. Design an HLD for an Online Examination System applying these principles.
Ans.
(Provided in Draw.io file)