7th

1.c

Compare and contrast the Incremental Model and Spiral Model..in table

Here’s a clear comparison of the **Incremental Model** and **Spiral Model** in a table format:

Aspect	**Incremental Model**	**Spiral Model**
Definition	Develops the system in small, incremental parts or builds.	Combines iterative development with systematic risk analysis.
Approach	Linear sequence of increments; each adds functional features.	Cyclical process of repeated spirals (phases) with risk assessment.
Phases	Requirements, Design, Implementation, and Testing repeat per increment.	Each spiral includes Planning, Risk Analysis, Engineering, and Evaluation.
Focus	Functionality delivered in parts; feedback after each increment.	Focus on risk management at every stage; high emphasis on risk analysis.
Risk Handling	Basic risk management; not its main focus.	Explicitly identifies and reduces risks in each spiral loop.
Customer Interaction	Customer feedback after each increment; requirements evolve.	Continuous customer involvement at every spiral cycle.
Flexibility	Moderate flexibility for changes between increments.	Highly flexible; changes are easier to handle due to iterative spirals.
Cost & Time	Less costly than Spiral for small to medium projects.	Can be expensive and time-consuming due to thorough risk analysis.
Complexity	Simpler to manage and understand.	More complex due to risk analysis and multiple iterations.
Best suited for	Well-defined, less risky projects needing staged delivery.	Large, complex, high-risk projects with unclear requirements.

Aspect	Incremental Model	Spiral Model
Examples	Business applications, simple software products.	Critical systems, aerospace, large financial systems.

Let me know if you'd like me to explain this comparison in simple language or provide an example!

Highlight the key differences in their approach to development, risk management, and adaptability

to changing requirements.

Discuss scenarios where one model might be more suitable than the other.(increment and spiral model)

Alright! Let's break this down step by step — highlighting the **key differences** in **development approach**, **risk management**, and **adaptability**, and then I'll explain **when to use which model** with simple examples.

🔑 Key Differences

Aspect	Incremental Model	Spiral Model
Development Approach	Develops the system in small, planned parts (increments). Each increment adds a piece of the final system. Earlier increments may deliver core features, later ones add enhancements.	Develops the system through repeated spirals (cycles). Each cycle includes planning, risk analysis, building, and evaluating. The system gradually grows with constant feedback.
Risk Management	Risk management is minimal or basic. Assumes requirements are mostly stable.	Strong focus on risk identification and mitigation in every cycle. Helps handle uncertain, high-risk projects.
Adaptability to Changing Requirements	Moderate adaptability. Changes can be added in later increments, but big changes might affect earlier builds.	Highly adaptable. Changes are expected and easily handled in each new spiral cycle. The model encourages revisiting requirements.

✅ When to Use Which Model

Scenario	Incremental Model is Better When…	Spiral Model is Better When…
Project Size & Complexity	Small to medium projects with well-understood requirements. Example: A simple business website where you first launch the homepage, then add features like a blog, user login, or payment later.	Large, complex, high-risk projects with uncertain requirements. Example: Developing a new operating system, an aerospace control system, or a banking system where risks must be carefully analyzed.
Requirements Stability	Requirements are mostly known upfront and unlikely to change drastically.	Requirements are unclear at the beginning and expected to evolve.
Risk Level	Low-risk projects where formal risk analysis is not critical.	High-risk projects where managing and reducing risk is vital to success.
Budget & Time	Limited budget and shorter timelines.	Flexible budget and time — risk analysis takes extra time and cost.

⚡ Simple Example

- **Incremental:** Making a mobile app for a local shop. First release: basic shopping and checkout. Next update: user reviews. Next update: loyalty points. Requirements don't change much.
- **Spiral:** Building a defense system for an aircraft. Each cycle checks for new risks (technical, safety, legal), tests new designs, and improves. Requirements change with new technology and feedback.

✅ Key takeaway:

- **Incremental** = step-by-step delivery with moderate changes and low risk.
- **Spiral** = handles changing, risky projects where constant checking and improvement are necessary.

If you'd like, I can also create a short chart or flow diagram — just say *yes*!

6th

1

b)Define agile and scrum in software engineering process model. Write down their procedures along with scenario when the processes are suitable.

Alright! Let’s break this down step by step in simple, clear language:

✅ Definition

Agile

Agile is a **software development process model** that focuses on **flexibility**, **continuous feedback**, **collaboration**, and **iterative development**.

- Software is developed **in small, manageable parts** (called *iterations* or *sprints*).
- The team adapts quickly to **changing requirements**, customer feedback, and unexpected issues.
- Popular Agile methods include **Scrum**, **Kanban**, **Extreme Programming (XP)**, etc.

Scrum

Scrum is one of the **most popular Agile frameworks**.

- It provides **specific roles**, **events**, and **artifacts** to organize the work.
- Work is done in **time-boxed iterations** called **Sprints** (usually 2–4 weeks).
- It promotes **teamwork**, **transparency**, and **continuous improvement**.

✅ Procedures

Agile Process Procedure (General)

1️⃣ **Requirement Gathering:**
Requirements are gathered at a high level; detailed requirements evolve over time.

2️⃣ **Planning:**
Plan short iterations (1–4 weeks).

3️⃣ **Design & Development:**
Develop a small piece of the product.

4️⃣ **Testing:**
Test the small part immediately.

5️⃣ **Review & Feedback:**
Get feedback from customers or stakeholders.

6️⃣ **Iteration:**
Use feedback to improve or add new features in the next iteration.

7️⃣ **Release:**
After several iterations, deliver the final product.

Scrum Process Procedure

1️⃣ **Product Backlog Creation:**
Make a list of all features (Product Backlog) prioritized by the **Product Owner**.

2️⃣ **Sprint Planning:**
The team selects what to do in the next **Sprint** from the Product Backlog.

3️⃣ **Sprint:**
The team works for **2–4 weeks** to complete the selected tasks.

4️⃣ **Daily Scrum (Daily Standup):**
A short meeting (15 mins) every day to discuss:

- What was done yesterday?
- What will be done today?
- Any obstacles?

5️⃣ **Sprint Review:**
At the end of the Sprint, the team shows the work done to stakeholders.

6️⃣ **Sprint Retrospective:**
The team discusses:

- What went well?
- What can be improved?

- What actions to take next Sprint?

7️⃣ **Repeat:**
Start the next Sprint with updated priorities.

✅ When are Agile & Scrum Suitable?

Situation	Agile	Scrum
Requirements keep changing	✅ Suitable	✅ Very suitable
High customer involvement	✅ Suitable	✅ Suitable
Small to medium-sized teams	✅ Suitable	✅ Suitable
Need for quick delivery	✅ Suitable	✅ Suitable
Complex projects with uncertain requirements	✅ Suitable	✅ Suitable
Large, stable projects with fixed requirements	❌ Not ideal	❌ Not ideal
Safety-critical systems (e.g., medical, aerospace)	❌ Not preferred	❌ Not preferred

✅ Example Scenario

Example:
A company wants to build a **mobile banking app**.

- The market changes quickly, so they need to add new features often.
- Users give feedback that changes requirements.
- The development team uses **Scrum** to work in **2-week Sprints**.
- Every Sprint, they release new updates, get user feedback, and adjust the next Sprint's plan.

In short:

- **Agile** is the **mindset and approach** for flexible, iterative development.

- **Scrum** is a **specific Agile method** with clear rules, roles, and time-boxed work cycles.

If you'd like, I can also make a simple table or diagram for this. Just say **yes!** 📌

c)

What is software engineering? Is it applicable when WebApps are built? If so, how it might be modified to accommodate the unique characteristics of WebApps? [3]

Alright! Let's break this down step by step so it's clear and easy to understand.

✅ What is Software Engineering?

Software Engineering is the **systematic, disciplined, and structured approach** to the **development, operation, and maintenance of software**.
👉 It uses **engineering principles** to build reliable, efficient, maintainable, and cost-effective software systems.
It includes:

- Requirements analysis
- Design
- Development (coding)
- Testing
- Deployment
- Maintenance

✅ Is it applicable for WebApps?

Yes!
Software Engineering **is very much applicable** when building **Web Applications (WebApps)**.

- WebApps are also software systems, just delivered via the web.
- They need to be planned, designed, developed, tested, deployed, and maintained — all core tasks of Software Engineering.

✅ How might Software Engineering be modified for WebApps?

WebApps have **unique characteristics**, so the traditional Software Engineering approach may need **some modifications**, such as:

Unique Characteristics of WebApps	**How Software Engineering is modified**
Frequent updates and rapid delivery	Use **Agile** or **Scrum** for fast iterations and continuous deployment.
Wide range of users & devices	Focus on **responsive design**, **cross-browser testing**, and **performance optimization**.
Security risks	Apply strong **web security engineering practices** (e.g., HTTPS, input validation, authentication).
24/7 availability	Design for **high availability**, **scalability**, and **reliability**.
User experience is key	Integrate **UX/UI design** principles into the process.
Integration with other services	Plan for **API design**, **third-party services**, **cloud hosting**, etc.

✅ **In summary (for 3 marks)**:

Software Engineering is a structured approach to develop software systematically. It **definitely applies** to WebApps but is often adapted with **Agile methods**, **focus on usability**, **security**, **continuous delivery**, and **cross-platform compatibility** to handle the unique demands of web-based systems.

If you'd like, I can help you rewrite this in an even shorter exam-style answer — just say **yes!** 📚✨

d. Why software project planning and tracking is necessary in software engineering? [2]

✅ Why is software project planning and tracking necessary in software engineering? [2 Marks]

Software project planning is needed to:

- **Define what to do, how to do it, when to do it, and who will do it.**
- Allocate resources, estimate cost and time, and identify risks.

Project tracking is needed to:

- **Monitor progress** to see if the project is on schedule and within budget.
- Detect problems early and take corrective actions to ensure successful delivery.

👉 Together, they help deliver **high-quality software on time and within budget**.

2.

A)

What do you mean by Risk analysis and management? What steps are involved in it. Explain in details. [3]

Alright! Here’s a clear and complete answer for **Risk Analysis and Management** with steps — perfect for a **3-mark** question.

✅ What is Risk Analysis and Management?

Risk Analysis and Management is a **systematic process** in **software engineering** to:

- **Identify possible risks** that may affect the project.
- **Analyze** how serious they are.
- **Plan** how to **minimize or handle** them.

Goal: Reduce problems that could **delay the project**, **increase cost**, or **lower quality**.

✅ Steps involved in Risk Analysis and Management

There are **three main steps**:

1️⃣ Risk Identification

- **What it means:** Find out **what could go wrong** in the project.
- **Examples of risks:**
    - Unrealistic deadlines
    - New technology failure

  - Key team member leaving
  - Security issues
  - Changing customer requirements

2️⃣ Risk Analysis (Risk Assessment)

- **What it means:** Study each risk to see:
  - **Probability:** How likely is it to happen? (Low, Medium, High)
  - **Impact:** How bad will it be if it happens? (Minor, Major, Critical)
- **Purpose:** Helps **prioritize** which risks need more attention.

3️⃣ Risk Management (Risk Control/Planning)

- **What it means:** Plan how to **reduce, avoid, or handle** the risks.
- **Common strategies:**
  - **Avoid:** Change the plan to remove the risk.
  - **Reduce:** Take steps to lower the chance or impact.
  - **Accept:** If it's minor, prepare to deal with it.
  - **Monitor:** Keep an eye on it throughout the project.
- Example: If there's a risk of losing a key developer, have backup staff or extra training.

✅ In summary

Risk Analysis and Management means **finding, studying, and handling possible problems** to **increase project success**.
It involves **Identification**, **Analysis**, and **Management/Control** of risks.

If you'd like, I can help you turn this into a short bullet-point version for quick memorization — just ask! 🌟

b)

) The requirements engineering process is accomplished through the execution of seven distinct functions. Specifically, requirements engineering encompasses requirements inception, analysis, specification, verification, and management. Briefly describe these activities. [3]

✅ Requirements Engineering Process — Main Activities [3 Marks]

Requirements Engineering involves **systematically finding out, defining, and managing** what the software should do. The key activities are:

1️⃣ Requirements Inception (or Elicitation)

- **What:** Gather requirements from stakeholders (customers, users, managers).
- **How:** Interviews, surveys, workshops, observation.
- **Goal:** Understand what the user **really wants and needs**.

2️⃣ Requirements Analysis

- **What:** Study and **refine gathered requirements**.
- **How:** Detect conflicts, overlaps, missing info.
- **Goal:** Make sure requirements are **complete, clear, and feasible**.

3️⃣ Requirements Specification

- **What:** Document the requirements **formally** in a **Software Requirements Specification (SRS)**.
- **How:** Use clear language, diagrams, models.
- **Goal:** Provide a **clear, agreed-upon reference** for developers and testers.

4️⃣ Requirements Verification and Validation

- **What:** Check that the requirements are **correct and agreed by stakeholders**.
- **How:** Reviews, walkthroughs, prototyping.
- **Goal:** Ensure requirements **match user needs and are error-free**.

5️⃣ Requirements Management

- **What: Track changes**, updates, and versions of requirements throughout the project.
- **How:** Use tools and processes to handle **requirement changes**.
- **Goal:** Keep the project aligned when requirements **evolve or new needs arise**.

✅ In summary:

Requirements Engineering includes: **Inception → Analysis → Specification → Verification → Management** to make sure **the right software is built correctly**.

Let me know if you'd like a short bullet or diagram version! 📌 ✨

c) Define people, process, and product. Explain the process quality and product quality in details. [3]

✅ People, Process, and Product — Definition & Quality Explanation [3 Marks]

✅ Definition

🔹 People

- The **individuals and teams** involved in developing the software — e.g., developers, testers, project managers, designers, customers, users.
- People are the **most important asset** in software engineering.

🔹 Process

- The **framework of tasks, methods, and practices** used to develop and maintain software.
- A good process guides **how work is planned, done, and checked**.

🔹 Product

- The **final software system** and its related deliverables (code, documents, manuals, reports).
- The **output** of the software engineering effort.

✅ Process Quality

- **Process Quality** means how **effective and efficient** the **software development process** is.
- A **high-quality process** reduces mistakes, improves productivity, and produces better products.
- Example: Using standard practices (like reviews, testing, documentation) to **ensure work is done properly**.
- Well-defined processes (like **ISO 9001**, **CMMI**) help achieve consistent results.

✅ Product Quality

- **Product Quality** means how well the **final software** meets **customer needs**, **requirements**, and **expectations**.
- Key factors: correctness, reliability, usability, performance, security, maintainability.
- A **high-quality product** is **bug-free**, **easy to use**, performs well, and is easy to maintain or update.

✅ In short:

People build the software, the **process** guides how they build it, and the **product** is the result.
Good process quality helps deliver **high product quality**.

If you'd like, I can write this in an even shorter exam-style version — just say **yes!** 🌟

3.
a) Define design pattern. What do you mean by coupling and cohesion? [2]

b) Suppose you are working on a user interface toolkit and you wish to support adding borders and scroll bars to windows. Also suppose that scroll bars on the windows can either be horizontal or vertical. Now, draw the UML for the scenario. [3]

c) Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
Which design pattern is used to address the scenario? Write down the UML and implementation of the design pattern. [3]

d) Why simulation design pattern is used? Write down the reasons along with its application. [2]

Alright — let's break this whole **question 3** into clear parts, with straightforward explanations and examples.

✅ 3(a)

Definition of Design Pattern

A **design pattern** is a **general, reusable solution** to a **common problem** that occurs in software design.
👉 It is **not code**, but a **template or guide** for solving a design issue in a standard way.

Coupling and Cohesion

- **Coupling:**
  👉 Coupling means the **degree of dependency** between different modules/classes.
    - **Low coupling** is good — modules are **independent**, easier to change.
    - **High coupling** is bad — modules depend too much on each other.
- **Cohesion:**
  👉 Cohesion means how **closely related and focused** the **functions inside a single module** are.
    - **High cohesion** is good — the module does **one clear task**.
    - **Low cohesion** is bad — the module does **many unrelated tasks**.

✅ **Goal: High cohesion, low coupling.**

✅ 3(b)

Scenario: Borders & Scrollbars for Windows

This is a classic example of the **Decorator Pattern**, which allows adding features **dynamically**.

📌 UML for the Decorator Pattern in this scenario:

```
+----------------+      +-----------------+
|  Component    |<------+  Decorator     |
| (Window)      |      | (WindowDecorator)|
+----------------+      +-----------------+
       ^                    ^
       |                    |
+----------------+      +----------------+
| ConcreteComponent |     | ConcreteDecorator |
|  (SimpleWindow)  |     | (BorderDecorator)  |
+----------------+      +----------------+
                           |
                   +----------------+
                   | (ScrollbarDecorator) |
                   +----------------+
```

✅ Explanation:

- **Component:** Defines the interface (e.g., Window).
- **ConcreteComponent:** Basic window without decorations (e.g., SimpleWindow).
- **Decorator:** Has a reference to a Window and adds extra behavior.
- **ConcreteDecorator:** Adds borders or scrollbars.

So you can **wrap** a window with a BorderDecorator and then a ScrollbarDecorator (for horizontal or vertical).

✅ 3(c)

Scenario: Request handled by a chain

Design Pattern: 👉 Chain of Responsibility

📌 UML for Chain of Responsibility:

```
+-----------------+
|  Handler        |
| +handleRequest()|
+-----------------+
        ^
        |
+------------------+
|  ConcreteHandler |
| +handleRequest() |
+------------------+
        |
  [ next handler ]
```

✅ Explanation:

- **Handler:** Defines an interface for handling requests.
- **ConcreteHandler:** Implements request handling, forwards to next handler if needed.

Example Implementation (pseudo-code)

```
class Handler:
    def __init__(self, next_handler=None):
        self.next = next_handler

    def handle(self, request):
        if self.next:
            self.next.handle(request)


class ConcreteHandlerA(Handler):
    def handle(self, request):
        if request == "A":
            print("Handled by A")
        elif self.next:
            self.next.handle(request)


class ConcreteHandlerB(Handler):
    def handle(self, request):
        if request == "B":
            print("Handled by B")
        elif self.next:
            self.next.handle(request)


# Example chain
handler_chain = ConcreteHandlerA(ConcreteHandlerB())
handler_chain.handle("B")  # Output: Handled by B
```

✅ 3(d)

Why Simulation Design Pattern is used?

Purpose:

- To **model real-world systems** or **complex behaviors** by imitating how they work.
- Helps to **analyze performance**, **test scenarios**, and **predict outcomes** without building the actual system.

Reasons:

- Saves **time and cost** by testing ideas virtually.
- Allows **safe experiments** with risky or expensive systems.
- Supports **training** and **decision-making**.

Application:

- Used in **scientific modeling**, **traffic systems**, **network simulations**, **flight simulators**, and **game development**.

✅ Short answer:

Simulation design pattern is used to **mimic real-world systems** for testing, prediction, and training without real-world risks or costs.

Let me know if you'd like all of this compiled as **neat exam notes or a PDF!** 📚 ✨

4.
a) What do you mean by Code Smells and Refactoring? [3]

b) Write down the following code smells problems with solutions
i. Feature Envy ii. Middle man iii. Shotgun Surgery iv. Refused Bequest v. Long method [5]

c) A project has 10 modules. Regression cases 80 nos, test cases executed during regression testing find the percentage of test case executed. [2]

d) Write down the comparisons among Black Box, White Box, and Gray Box Testing. [3]

Alright! Let's break down **Question 4** step by step with clear, short answers — perfect for an exam.

✅ 4(a)

Code Smells

- **Definition:**
  A **code smell** is a **sign of potential problems** in the source code that **may cause bugs or**

make maintenance harder.
👉 They are **not bugs**, but **bad design choices** that reduce code quality.

Examples: Long classes, duplicate code, too many responsibilities.

Refactoring

- **Definition:**
  Refactoring means **improving the structure** of the existing code **without changing its behavior**.
  👉 Goal: Make code **cleaner, simpler, easier to maintain**.

Example: Splitting a long method into smaller methods.

✅ **4(b)**

Code Smells Problems & Solutions

Code Smell	Problem	Solution
i. Feature Envy	A method **uses data from another class** more than its own class.	Move the method to the class it uses most (use **Move Method** refactoring).
ii. Middle Man	A class **does nothing except delegate work** to another class.	Remove the middle class and let clients communicate directly (**Remove Middle Man**).
iii. Shotgun Surgery	A **small change** in one place needs **many small changes in many other classes**.	Use **Move Method** or **Extract Class** to group related code together.
iv. Refused Bequest	A subclass **doesn't want or use some methods** it inherits.	Use **Replace Inheritance with Delegation** — use composition instead of inheritance.
v. Long Method	A method does **too much work**, making it hard to read or reuse.	Break it into smaller methods (**Extract Method**) with clear names.

✅ **4(c)**

Given:

- Modules: 10 (not needed for this calculation)
- Regression test cases: 80
- Test cases executed during regression: ? (Not given, but let’s assume **all 80 executed**)

👉 **Percentage executed** = (Executed cases / Total cases) × 100%
If **all 80 were executed**:

Percentage=8080×100%=100%\text{Percentage} = \frac{80}{80} \times 100\% = 100\%

✅ **Answer: 100%**

(If you have a different number of executed cases, just plug it in.)

✅ **4(d)**

Comparison: Black Box vs White Box vs Gray Box Testing

Aspect	Black Box Testing	White Box Testing	Gray Box Testing
Knowledge of Code	No knowledge of internal code	Full knowledge of internal code	Partial knowledge of internal code
Who performs it	Testers, users	Developers	Testers with some development knowledge
Focus Area	Functionality, requirements	Code structure, logic, internal paths	Functionality with partial internal understanding
Test Basis	Requirements & specifications	Design, code, algorithms	High-level design + limited code info
Examples	System testing, Acceptance testing, UI testing	Unit testing, Path testing, Branch testing	Integration testing, Penetration testing, Security testing
Techniques Used	Equivalence Partitioning, Boundary Value Analysis	Statement coverage, Branch coverage, Condition coverage	Combination of black box and white box techniques

Aspect	Black Box Testing	White Box Testing	Gray Box Testing
Advantages	Unbiased, closer to user’s perspective, detects missing functions	Thorough, finds hidden bugs in code logic, optimizes code	Balanced approach, better test coverage, finds integration issues
Disadvantages	Cannot test internal code logic, may miss hidden bugs	Time-consuming, requires programming skills, cannot find missing functionality	May not cover full internal paths, still needs access to partial code
Level of Testing	High-level: System, Acceptance	Low-level: Unit, Integration	Both levels: System + some code checks
When used	When verifying that the software meets requirements	When testing critical code logic and paths	When tester has partial access to code and wants broader coverage

✅ **Summary:**

- **Black Box:** *What it does*
- **White Box:** *How it works*
- **Gray Box:** *A bit of both*

If you’d like, I can put all these answers into a neat **PDF** or **one-page note** — just say **yes!** 📚 ✨

Here are the **answers** to the questions from your uploaded image (Question 5):

5(a) What do you mean by testing? Do you think CMMI level is necessary? Justify your answer. [3]

Testing is the process of evaluating a software system or its components to find whether it meets the specified requirements and to identify any defects.

CMMI (Capability Maturity Model Integration) is a framework that helps organizations improve their software development processes.

✅ **CMMI is necessary** because:

- It ensures **standardized processes**.
- Improves **product quality and efficiency**.
- Helps in **reducing defects** and rework.
- Increases **customer satisfaction** and trust.

5(b) Write short notes on:

i. Reliability, Security, Downtime, Authentication

- **Reliability**: The ability of software to perform a required function under stated conditions for a specific time.
- **Security**: Protects software from unauthorized access, modification, and ensures data privacy.
- **Downtime**: The period when the system is not available or not functioning.
- **Authentication**: A process to confirm the identity of users (e.g., login with username & password).

ii. Error, Bug, Mistake, and Fault

- **Error**: Human action that produces an incorrect result.
- **Bug**: A flaw or defect in software causing incorrect or unexpected results.
- **Mistake**: Wrong step taken by the developer due to misunderstanding.
- **Fault**: Incorrect step, process, or data definition in a program.

5(c) Define basic path testing and test case. How to find out the cyclometric complexity? [2]

- **Basic Path Testing**: A white-box testing technique that uses the control flow graph to find all possible independent paths in the program and design test cases accordingly.
- **Test Case**: A set of conditions or steps to determine if a software feature works correctly.

Cyclomatic Complexity (CC) = $E - N + 2P$
Where:

- E = Number of edges
- N = Number of nodes
- P = Number of connected components (usually 1)

5(d) Define project management. Write down the qualities of a good project manager. [4]

Project Management is the process of planning, executing, and managing resources to complete specific goals within a defined timeline and budget.

✅ **Qualities of a Good Project Manager:**

1. **Leadership** – Able to guide and inspire the team.
2. **Communication Skills** – Clear communication with team and stakeholders.
3. **Problem Solving** – Quickly identifying and solving issues.
4. **Time Management** – Prioritizing tasks and meeting deadlines.
5. **Decision-Making** – Making timely and effective decisions.
6. **Adaptability** – Adjusting to changes and uncertainties.

Let me know if you want these answers in **Bangla** or **in short points for quick revision**.