**Software Development Life Cycle (SDLC)**

**What is SDLC?**

* SDLC means the steps to make software.
* Just like a story has a beginning, middle, and end — software making also has steps.
* These steps are called SDLC.

**What is Life Cycle Model?**

* It is like a picture or diagram that shows the steps of making software.
* It tells us what to do in each step.

**Why do we need SDLC?**

* If we don’t follow the steps, making software will be messy.
* The team must first choose the right model.
* Then, everyone works step by step, and s software is made in a proper way.

**Stage 1: Planning and Requirement Analysis**

* This is the **first and most important step** in SDLC.
* Here, the team talks with **clients, users, and experts** to understand what the software should do.
* The senior members of the team collect all the needs (requirements).
* They also **plan for quality** (how to make the software good) and **find possible risks** (what problems might happen).

👉 In simple words:  
This stage is like **making a shopping list before cooking**. If the list is wrong, the whole dish will be wrong.  
**Stage 2: Defining Requirements**

* After finding out what the client wants, the team writes it down clearly.
* These written needs are put into a special file called **SRS (Software Requirement Specification)**.
* The SRS document lists **everything the software should do**.
* The client or project owner checks and approves this document.

👉 In simple words:  
This stage is like **writing down the recipe after deciding what dish to cook**, so everyone agrees before starting.

**Stage 2: Defining Requirements**

* After knowing the needs, we **write them clearly**.
* This written file is called **SRS**.
* It shows **what the software must do**.
* The client checks and says **OK**.

👉 Simple: Write down the plan so everyone agrees.

**Stage 3: Designing the Software**

* After collecting and writing needs, we **plan how the software will look and work**.
* This design is made using the customer’s needs and requirement details.
* It works like a **blueprint** before building the real software.

👉 Simple: Make a plan (design) before building.

**tage 3: Designing the Software**

* Now we **make a plan** for the software.
* This plan shows **how it will look and work**.
* It is like a **drawing or map** before we build it.

👉 Simple: First think and draw, then build.

**Stage 4: Developing the Project**

* In this step, the **real coding starts**.
* Developers **write programs** using different coding tools.
* They follow rules so the code is **clean and correct**.
* This is where the **software is actually built**.

👉 Simple: Now we turn the plan into a **real working program**.

**Stage 5: Testing**

* After making the software, we **check it works or not**.
* We test every part and then the whole software.
* If there are mistakes (bugs), we fix them.

**Stage 5: Testing**

* After making the program, we **check if it works properly**.
* We test it to see if it **matches the needs** we wrote earlier.
* Different kinds of testing are done:
  + **Unit testing** (check small parts)
  + **Integration testing** (check parts together)
  + **System testing** (check the whole program)
  + **Acceptance testing** (final check if users are happy)

👉 Simple: This stage makes sure the **software works correctly** and solves the problem.

**Stage 6: Deployment**

* After checking everything, the software is **given to the users**.
* Users start using it in real life.
* Sometimes small improvements are added before release.
* After this, **maintenance** starts to keep it working well.

👉 Simple: Give the software to people and make sure it **keeps working properly**.

**Stage 7: Maintenance**

* After users start using the software, **problems may appear**.
* Sometimes small changes or updates are needed.
* Taking care of the software and fixing problems is called **maintenance**.

👉 Simple: Keep the software **working well all the time**.

**all 7 SDLC stages in one super simple, easy-to-read format** for your presentation:

**SDLC – Easy Version**

**Stage 1: Planning & Requirement Analysis**

* Talk with users and experts to know what is needed.
* Make a plan and check for risks.
* Simple: **Make a list before starting.**

**Stage 2: Defining Requirements**

* Write down everything the software must do (SRS).
* Get approval from the client.
* Simple: **Agree on the plan first.**

**Stage 3: Designing**

* Plan how the software will look and work.
* Make a blueprint before building.
* Simple: **Draw before building.**

**Stage 4: Developing / Coding**

* Write the actual code using tools and rules.
* Simple: **Turn the plan into a working program.**

**Stage 5: Testing**

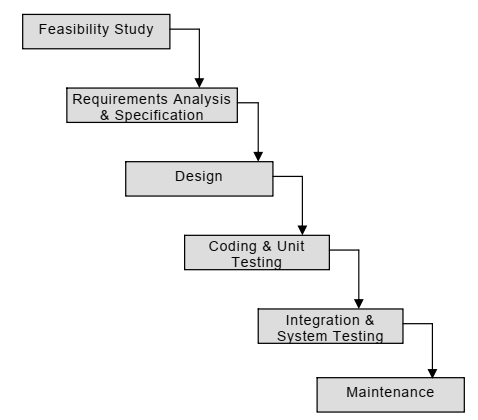
* Check if the software works correctly.
* Fix mistakes (bugs).
* Simple: **Make sure it works perfectly.**

**Stage 6: Deployment**

* Give the software to users.
* Add small improvements if needed.
* Simple: **Give it to people to use.**

**Stage 7: Maintenance**

* Fix problems and update the software after users start using it.
* Simple: **Keep it working well.**



**SDLC Models – Easy Version**

**What is SDLC Model?**

* A SDLC model shows the **steps to make software**.
* It tells developers **what to do first, second, and so on**.
* Steps do not overlap; you finish one before starting the next.

A diagram of a diagram

Description automatically generated

**Waterfall Model**

* Named “Waterfall” because its steps **look like a waterfall** in a diagram.
* Steps go **top to bottom**, one after another.
* There are two types:
  1. **Classical Waterfall Model**
  2. **Iterative Waterfall Model**

**Waterfall Model – Slide Version**

1. **Feasibility Study: Check if project is possible and worth it.**
2. **Requirements: Write exactly what software should do (SRS).**
3. **Design: Plan software and draw blueprint (SDD).**
4. **Implementation & Unit Testing: Write code and test small parts.**
5. **Integration & System Testing: Combine parts and test full software.**
6. **Operation & Maintenance: Deliver software and fix/update when needed.**

**When to use Waterfall Model?**

* Requirements **don’t change**.
* Projects are **short and simple**.
* Tools and technology **stay the same**.
* Resources are **ready and available**.

**Advantages of Waterfall Model**

* Easy to **understand and use**.
* Requirements are **clear and fixed**.
* Each phase has a **start and end point**.
* You can **know the cost and time** before starting.
* Easy for **customers to track progress**.

**Disadvantages of Waterfall Model (Easy Version)**

* **High risk** – not good for big or complicated projects.
* **Changes are hard** – you cannot easily change requirements once coding starts.
* **Hard to go back** – if you are in coding and need to change something, it’s difficult.
* **Late testing** – problems are found only at the end, so it’s harder to plan for risks.

**What is RAD?**

* RAD is a **fast way to make software**.
* It is a step-by-step (linear) process but focuses on **building quickly using small parts**.

**How RAD works:**

1. **Gather requirements** – Talk with users using workshops or groups.

**2. Prototyping** – Make early versions and test them with users.

**3. Reuse components** – Use existing software pieces to save time.

👉 Simple: **Build small parts fast, test them, and use ready pieces** to make the software quicker and better.

**AD Extra Points (Easy Version)**

* The schedule is **fast and strict**, some improvements wait for the next version.
* Team talks and reviews are **less formal**, more flexible.

👉 Simple: **Quick work, easy team talks, small changes can wait for later.**

**RAD – Rapid Application Development**

**Phases:**

1. **Business Modelling** – Learn what the business does and how data moves.
2. **Data Modelling** – Organize the data and show connections.
3. **Process Modelling** – Plan how data is used and changed.
4. **Application Generation** – Build software fast using tools.
5. **Testing & Turnover** – Test new parts and check everything works.

**When to Use:**

* Project is **small and fast** (2–3 months).
* Requirements are **clear**.
* Low technical risk.
* Tools are available for **automatic coding**.

**Advantages:**

* Fast development.
* Reuse parts to save time.
* Users can test early.

**Disadvantages:**

* Not for big projects.
* Needs skilled developers and tools.
* Can be expensive.

**V-Model – Easy Version**

**What is it?  
Make software step by step and test along the way. Left side = develop, Right side = test.**

**Steps (Phases)**

**Development:**

1. **Requirement: Decide what software should do.**
2. **System Design: Plan how it will work.**
3. **High-Level Design: Divide into parts.**
4. **Low-Level Design: Plan details of each part.**
5. **Coding: Write the code.**

**Testing:**

1. **Unit Test: Test each part.**
2. **Integration Test: Test combined parts.**
3. **System Test: Test whole software.**
4. **Acceptance Test: Client checks and accepts it.**

**When to Use:**

* **Requirements are clear.**
* **Small or medium projects.**
* **Testing is important.**

**Advantages:**

* **Easy to find problems.**
* **Each step is clear.**

**Disadvantages:**

* **Hard to change things.**
* **Expensive for big projects.**

**What is Spiral Model?**

* Make software in **loops (spirals)**.
* Each loop **improves the software** step by step.
* Focuses on **risk management** and planning.

**Each Loop has 4 Steps:**

1. **Objective** – Decide what to do in this loop.
2. **Risk Check** – Find problems or risks and plan how to reduce them.
3. **Build & Test** – Make the software and test it carefully.
4. **Plan Next** – Plan the next loop based on feedback.

**When to Use:**

* Big or tricky projects.
* Things can **change during development**.
* Risk is **high**.
* Users want to **see progress in stages**.

**Advantages:**

* Finds problems **early**, reduces risk.
* Works well for **big and complex projects**.
* Users can **give feedback** in each loop.
* Flexible – can **handle changes** in requirements.

**Disadvantages:**

* Can be **costly** due to repeated loops.
* Needs **skilled people** to manage risks and loops.
* Planning is **complicated**.
* Can take **longer time** if loops are many.

**Increment** modal;  
**What is it?**  
Build software in small parts (increments) step by step. Each part works on its own.

**Phases / Steps (4 main steps)**

* **Requirement:** Decide what features are needed for this increment.
* **Design:** Plan how the module will work and connect with other parts.
* **Development / Implementation:** Write the code for the module.
* **Testing:** Check the module for errors and fix them.
* Repeat these steps for each increment until the full system is complete.

**When to Use:**

* Requirements are clear.
* Need working parts early.
* Can deliver software step by step.

**Advantages:**

* Users can use parts early.
* Easy to test small modules.
* Problems are easier to find and fix.
* Flexible for later changes.
* **Disadvantages:**
* Integration of all modules at the end may be tricky.
* Planning all increments in advance can be hard.
* Not good if modules heavily depend on each other.

**Agile Model – Easy Version (Expanded)**

**What is Agile?**

* Agile is a **flexible and fast way** to make software.
* Work is done in **small parts called sprints**.
* Focuses on **team collaboration, user feedback, and continuous improvement**.

**Phases (Steps in a Sprint)**

* **Plan** – Decide what to do next.
* **Design** – Make a simple plan.
* **Build** – Make the feature.
* **Test** – Check it works.
* **Show** – Show to users.
* **Fix** – Make changes if needed.
* **Repeat** – Do the next small part.

**Agile Methods (Ways to Work)**

* **Scrum** – Work in sprints, have short daily meetings.
* **Kanban** – Use a board: To Do → Doing → Done.
* **XP (Extreme Programming)** – Test a lot and release often.
* **Lean** – Work fast and avoid waste.
* ✅ Simple:
* Agile is **fast, flexible, and user-friendly**.
* Work in **small steps**.
* Use **Scrum, Kanban, XP, or Lean** to stay organized.

**How Agile Works:**

1. **Plan Sprint** – Decide what small feature to build next.
2. **Develop & Test** – Build the feature and test it.
3. **Review** – Show it to users and get feedback.
4. **Improve** – Make changes and prepare for the next sprint.
5. Repeat these steps until the full project is done.

**When to Use Agile:**

* Requirements **change frequently**.
* Users want to **see progress quickly**.
* Projects are **complex or large**.
* Fast delivery is needed.

**Advantages of Agile:**

* **Flexible** – easy to adapt to changes.
* **User feedback** is included early.
* Problems are **found and solved early**.
* Works well for **big and small projects**.
* **Team collaboration** is strong.

**Disadvantages of Agile:**

* Needs **skilled and cooperative team**.
* Hard to **predict total cost and time**.
* Requires **continuous communication**.
* Can be **chaotic** if team or planning is weak.

**কোনো SDLC মডেল কবে ভালো:**

1. **Waterfall**

* ভালো যখন: প্রজেক্ট ছোট, সরল, এবং চেঞ্জ খুব কম হয়।
* সুবিধা: সহজ, সব কিছু আগে থেকে ঠিক করা যায়।
* অসুবিধা: চেঞ্জ করা কঠিন, বড় প্রজেক্টে ঠিকমতো কাজ হয় না।

1. **RAD (Rapid Application Development)**

* ভালো যখন: প্রজেক্ট **দ্রুত তৈরি করতে হবে** (২-৩ মাস), এবং প্রয়োজন স্পষ্ট।
* সুবিধা: দ্রুত, পুরানো কোড ব্যবহার করা যায়।
* অসুবিধা: বড় প্রজেক্টে ঠিকমতো কাজ করে না, খরচ বেশি হতে পারে।

1. **Spiral**

* ভালো যখন: **বড় বা জটিল প্রজেক্ট**, এবং চেঞ্জ আসতে পারে।
* সুবিধা: ঝুঁকি কম হয়, সমস্যা আগেই ধরা যায়।
* অসুবিধা: খরচ বেশি, পরিকল্পনা জটিল।

1. **Agile**

* ভালো যখন: চেঞ্জ হতে পারে, **ইউজার ফিডব্যাক দরকার**, দ্রুত রেজাল্ট দিতে হবে।
* সুবিধা: খুবই নমনীয়, দ্রুত উন্নতি হয়, ইউজার ফিডব্যাক পাওয়া যায়।
* অসুবিধা: স্কিলফুল টিম লাগে, খরচ ও সময় ঠিক বলা কঠিন।

**সারাংশ:**

* ছোট ও সহজ প্রজেক্ট → **Waterfall**
* দ্রুত ও ছোট প্রজেক্ট → **RAD**
* বড় ও ঝুঁকিপূর্ণ প্রজেক্ট → **Spiral**
* চেঞ্জ বেশি, ইউজার ফিডব্যাক চাই → **Agile  
  ১. Waterfall Model (জলপ্রপাত মডেল)**
* **চিনবেন কিভাবে:** ধাপগুলো **উপর থেকে নিচে ধারাবাহিকভাবে** চলে, এক ধাপ শেষ না হলে পরের ধাপ শুরু হয় না।
* **কবে ব্যবহার:** Requirements স্থির, ছোট বা সহজ প্রজেক্টে।
* **ফিচার:** সরল, স্পষ্ট ধাপ, testing শেষে।
* **২. Incremental Model (ধাপে ধাপে মডেল)**
* **চিনবেন কিভাবে:** সফটওয়্যারকে **ছোট ছোট অংশে ভাগ করে বানানো হয়**, প্রথমে কিছু ব্যবহারযোগ্য অংশ তৈরি হয়, পরে আরও যোগ করা হয়।
* **কবে ব্যবহার:** Requirements কিছুটা স্পষ্ট, দ্রুত কিছু ব্যবহারযোগ্য ফলাফল দরকার।
* **ফিচার:** ব্যবহারযোগ্য অংশ প্রথমে, নতুন ফিচার পরে যোগ।
* **৩. Spiral Model (স্পাইরাল / লুপ মডেল)**
* **চিনবেন কিভাবে:** **লুপ বা স্পাইরাল আকারে** সফটওয়্যার তৈরি হয়, প্রতি লুপে পরীক্ষা ও ঝুঁকি বিশ্লেষণ করা হয়।
* **কবে ব্যবহার:** বড়, জটিল বা ঝুঁকিপূর্ণ প্রজেক্টে।
* **ফিচার:** ঝুঁকি কমানো সহজ, ব্যবহারকারী progress দেখতে পারে।
* **৪. RAD Model (Rapid Application Development)**
* **চিনবেন কিভাবে:** **দ্রুত প্রোটোটাইপ বানানো হয়**, ছোট টিম এবং পুনঃব্যবহারযোগ্য কম্পোনেন্ট ব্যবহার করা হয়।
* **কবে ব্যবহার:** Requirements জানা আছে, দ্রুত software deliver করতে হবে।
* **ফিচার:** দ্রুত তৈরি, পুনঃব্যবহার, ব্যবহারকারীর feedback তাড়াতাড়ি।
* **৫. Agile Model (অ্যাজাইল মডেল)**
* **চিনবেন কিভাবে:** **ছোট ছোট iteration / sprint** করে software তৈরি হয়। Feedback পাওয়া যায় প্রতি ধাপে।
* **কবে ব্যবহার:** Requirements পরিবর্তনশীল, দ্রুত market release দরকার।
* **ফিচার:** flexible, user feedback অনুযায়ী উন্নতি, দ্রুত delivery।
* **৬. V-Model (Verification & Validation Model)**
* **চিনবেন কিভাবে:** **V আকারে ধাপ**, বাম দিকে development, ডান দিকে testing।
* **কবে ব্যবহার:** Requirements স্পষ্ট, testing গুরুত্বপূর্ণ।
* **ফিচার:** প্রতিটি development ধাপের সাথে corresponding test থাকে।
* 💡 **স্মরণে রাখার ট্রিক:**
* **Waterfall:** ধাপে ধাপে, একদিকে উপরে নিচে।
* **Incremental:** ধাপে ধাপে ছোট ছোট অংশ।
* **Spiral:** লুপ / ঝুঁকি যাচাই।
* **RAD:** দ্রুত prototyping।
* **Agile:** iteration / sprint, flexible।
* **V-Model:** V আকারে development + testing।
* **Scenario 1 – Waterfall**
* **Scenario:**  
  একটি স্কুলের জন্য ছোট একটা অ্যাপ বানাতে হবে। সব ফিচার আগেই ঠিক আছে, চেঞ্জ হওয়ার সম্ভাবনা নেই।
* **Question:** কোন SDLC মডেল সবচেয়ে ভালো হবে?  
  **Answer:** Waterfall  
  **Reason:** প্রজেক্ট ছোট ও সহজ, requirements fixed।
* **Scenario 2 – RAD**
* **Scenario:**  
  একটি দোকানের জন্য ২ মাসের মধ্যে সেলস অ্যাপ বানাতে হবে। ফাংশনগুলো ঠিক জানা আছে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** RAD  
  **Reason:** দ্রুত তৈরি করতে হবে, modular & reusable parts ব্যবহার করা যায়।
* **Scenario 3 – Spiral**
* **Scenario:**  
  একটি ব্যাংকের জন্য বড় সফটওয়্যার বানাতে হবে। ঝুঁকি বেশি, চেঞ্জ হতে পারে, জটিল প্রজেক্ট।
* **Question:** কোন মডেল ভালো হবে?  
  **Answer:** Spiral  
  **Reason:** বড় ও জটিল প্রজেক্টের জন্য, risk কমানো যায়।
* **Scenario 4 – Agile**
* **Scenario:**  
  একটি ওয়েব অ্যাপ বানানো হবে, প্রতি সপ্তাহে ইউজার ফিডব্যাক নেওয়া হবে, ফিচার পরিবর্তন হতে পারে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** Agile  
  **Reason:** নমনীয়, ইউজারের ফিডব্যাক অন্তর্ভুক্ত করা যায়, দ্রুত পরিবর্তন করা যায়।
* **Scenario 5 – Mixed/Comparison**
* **Scenario:**  
  একটি ছোট প্রজেক্ট বানাতে হবে। সময় কম, কিন্তু ইউজারের পরিবর্তন চাহিদা থাকতে পারে।
* **Question:** কোন মডেল ব্যবহার করা ভালো? Waterfall বা Agile?  
  **Answer:** Agile  
  **Reason:** সময় কম হলেও, Agile ব্যবহার করলে দ্রুত পরিবর্তন করা যায় এবং ইউজারের ফিডব্যাক নেওয়া যায়।
* **Scenario 6 – Waterfall**
* **Scenario:**  
  একটি লাইব্রেরির জন্য ছোট সফটওয়্যার বানাতে হবে। সব ফিচার আগেই ঠিক আছে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** Waterfall  
  **Reason:** ছোট প্রজেক্ট, requirements fixed।
* **Scenario 7 – RAD**
* **Scenario:**  
  একটি রেস্টুরেন্টের জন্য ৩ মাসের মধ্যে একটি অর্ডার অ্যাপ বানাতে হবে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** RAD  
  **Reason:** দ্রুত তৈরি করতে হবে, modular approach সহজ।
* **Scenario 8 – Spiral**
* **Scenario:**  
  একটি হাসপাতালের জন্য জটিল সফটওয়্যার বানাতে হবে। নিরাপত্তা ও risk বেশি।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** Spiral  
  **Reason:** বড়, ঝুঁকিপূর্ণ প্রজেক্টের জন্য, risk কমানো যায়।
* **Scenario 9 – Agile**
* **Scenario:**  
  একটি সোশ্যাল মিডিয়া অ্যাপ বানানো হবে। ইউজার feedback প্রতি সপ্তাহে নেওয়া হবে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** Agile  
  **Reason:** চেঞ্জ এবং ফিডব্যাক দ্রুত অন্তর্ভুক্ত করা যায়।
* **Scenario 10 – Waterfall**
* **Scenario:**  
  একটি ছোট কোম্পানির জন্য স্টক ট্র্যাকিং সফটওয়্যার বানাতে হবে। কোনো ফিচার পরিবর্তন হবে না।
* **Question:** কোন মডেল ব্যবহার করা ভালো?  
  **Answer:** Waterfall  
  **Reason:** Requirements fixed, project ছোট।
* **Scenario 11 – RAD**
* **Scenario:**  
  একটি স্কুলের জন্য ২ মাসের মধ্যে অ্যাটেনডেন্স অ্যাপ বানাতে হবে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** RAD  
  **Reason:** দ্রুত, modular, reusable parts ব্যবহার করা যায়।
* **Scenario 12 – Spiral**
* **Scenario:**  
  একটি ব্যাংকিং সফটওয়্যার যা সময়ের সাথে নতুন ফিচার যোগ হবে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** Spiral  
  **Reason:** বড় ও জটিল, risk management করতে সাহায্য করে।
* **Scenario 13 – Agile**
* **Scenario:**  
  একটি ওয়েব অ্যাপ বানানো হবে। ইউজারের feedback নিয়মিত নেওয়া হবে, চেঞ্জ থাকবে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** Agile  
  **Reason:** নমনীয়, দ্রুত চেঞ্জ করা যায়।
* **Scenario 14 – Waterfall**
* **Scenario:**  
  একটি ছোট প্রজেক্ট যেখানে সব requirement আগে থেকে ঠিক আছে।
* **Question:** কোন মডেল ভালো?  
  **Answer:** Waterfall  
  **Reason:** সহজ, fixed requirements।
* **Scenario 15 – RAD**
* **Scenario:**  
  একটি মোবাইল অ্যাপ ২–৩ মাসে বানাতে হবে।
* **Question:** কোন মডেল ব্যবহার করা উচিত?  
  **Answer:** RAD  
  **Reason:** দ্রুত বানানো যায়, modular approach।

aper A

Q1. Explain SDLC. List and describe its phases. (7 marks)

Answer:

SDLC (Software Development Life Cycle) means the step-by-step way of making software. It helps to plan, build, test and deliver software in a good way.

Phases:

1. Requirement: Talk with user, write down what they want.

2. Design: Make plan of system, draw diagrams, decide how software will look and work.

3. Coding: Write program based on design.

4. Testing: Check software, find errors and fix them.

5. Deployment: Give software to user, install and train them.

6. Maintenance: Repair bugs, update and improve software after release.

---

Q2. (VR project scenario)

i) Team structure (small team):

Project Manager → controls plan and time.

System Architect → plans software + hardware together.

Software Engineers (2) → make VR program and graphics.

Hardware Engineer → connect VR device with software.

UI/UX Designer → make it easy and comfortable for user.

Tester → check if system works well.

DevOps → help in building and releasing versions.

Why? Small but mixed team helps fast work. All skills are inside one group, so no delay.

ii) Best model: Agile with prototype (like Scrum + Spiral).

Work in short parts (sprints). Each sprint makes a small working feature.

Make quick prototypes to test VR speed and comfort.

Spiral idea helps to solve risky parts (like latency, sensors) early.

Why good?

Because project is risky and market is fast. Agile + prototype gives quick results, and Spiral helps remove big risks early.

---

Paper B

Q1. Define software engineering. Write needs. (5 marks)

Answer:

Software engineering means using rules and methods of engineering to make software in a careful and planned way.

Needs:

1. To control big and complex software.

2. To make software correct and good quality.

3. To finish in time and within budget.

4. To repair and update easily later.

5. To make users happy and safe.

---

Q2. Incremental vs Spiral Model (5 marks)

| **Point** | **Incremental Model** | **Spiral Model** |
| --- | --- | --- |
| Idea | Build software step by step | Work in loops, check risks, make prototype |
| Risk | Not directly handled | Risk is main focus |
| Result | Each step gives a small working part | Each loop gives plan or prototype |
| Use | Requirements clear, need fast delivery | Project is risky, big, or unclear |

---

Q3. (Genetic engineering project, R&D) (5 marks)

Answer:

I will use Spiral + Incremental model.

Spiral: Good for R&D because it checks risks and makes small prototype to test ideas.

Incremental: Good for delivering working parts step by step, so company gets product in one year.

First use Spiral to solve risky parts (new science, new methods).

Then use Incremental to add features (data, reports, UI) slowly.

Why: Because project is research-based (uncertain) but also time limited (1 year). This mix will manage both.