ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A)

(An AUTONOMOUS Institution)

Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada
Accredited by **NBA*** Accredited by **NAAC A+** Grade with CGPA of 3.40
Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P
Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

COURSE NAME: SOFTWARE ENGINERRING

COURSE CODE: 231CM5E01

QUESTION BANK

S.No.	Question	RBT level	CO	Marks
UNIT-I				
1a.	Describe the Notable changes in software development practices.	L2	CO1	5
b.	Discuss about evolutionary process models.	L1	CO1	5
2a.	Explain software development life cycle. Discuss various activities during SDLC.	L1	CO1	5
b.	Explain waterfall process model with merits and demerits.	L1	CO1	5
3a.	Explain the concept of Computer system engineering?	L2	CO1	5
b.	Draw and explain the spiral model and list the advantage and disadvantages.	L1	CO1	5
4a.	Explain about Exploratory style of Developing a software?	L2	CO1	5
b.	Explain about Agile Process Models and its principles.	L2	CO1	5
5a.	Discuss about Software development projects and its types?	L1	CO1	5
b.	Perform prototype analysis on “railway reservation” software.	L1	CO1	5
6a.	Differentiate between waterfall and incremental model	L2	CO1	5
b.	Summarize the Evolution of Software Engineering Methodologies.	L2	CO1	5
UNIT- II				
1a.	Explain metrics for project size estimation.	L3	CO2	5
b	Discuss the desirable characteristics of a good software requirement specification document.	L1	CO2	5
2a.	Describe Software project management complexities in detail.	L1	CO2	5
b.	List and explain various requirements elicitation techniques.	L1	CO2	5
3a.	Illustrate the responsibilities of a software project manager in detail.	L2	CO2	5
b.	What are the different categories of software development projects according to the COCOMO estimation model? Give an example of software product development projects belonging to each of these categories.	L2	CO2	5
4a.	Briefly explain various project size estimation techniques with an example and compare the advantages and disadvantages of expert judgement and Delphi technique.	L1	CO2	5

b.	Explain the differences between functional requirements and non-functional requirements by giving suitable examples.	L2	CO2	5
5a.	What is the goal of requirements analysis phase? Give reasons why the requirements analysis phase is a difficult one.	L1	CO2	5
b.	Explain how you can choose the best risk reduction technique when there are many ways of reducing a risk.	L2	CO2	5
6a.	What do you understand by an executable specification language? How is it different from a traditional procedural programming language? Give an example of an executable specification language.	L1	CO2	5
b	Discuss the relative advantages of formal and informal requirements specifications.	L2	CO2	5

SE MID-1 ANSWERS

UNIT - 1

1a. Describe the Notable changes in software development practices.

b. Discuss about evolutionary process models.

ANSWER:

Q1(a). Notable Changes in Software Development Practices

In the early days, software development was done in an ad-hoc way, without proper methods. As projects grew complex, many changes happened in practices.

Notable Changes are:

1. **From Ad-hoc to Structured Development**
   - Earlier coding was random and unplanned.
   - Now, software is developed in a systematic way using proper models and methodologies.
2. **From Control Flow to Structured Design**
   - Old programs were focused only on sequence, loops, and decisions.
   - Later, **structured design** and **data-flow design** gave better clarity and reduced complexity.
3. **From Procedure-Oriented to Object-Oriented Design**
   - Initially, focus was on functions and procedures.
   - Now, **object-oriented design** (classes, encapsulation, inheritance, polymorphism) is widely used.
4. **Use of High-Level Abstractions**
   - Earlier coding was in machine or assembly language.
   - Then came high-level languages (C, Java, Python) and frameworks, which make development faster and less error-prone.
5. **Component Reuse**
   - Earlier, programmers wrote everything from scratch.
   - Now, reusable modules, APIs, libraries, and design patterns are used to save cost and effort.
6. **Shift to Iterative and Agile Models**
   - Waterfall model was once popular but rigid.

   - Now, **Agile, Scrum, RAD, Incremental, Spiral** methods are used for flexibility and customer satisfaction.

7. **Focus on Quality and Maintenance**
   - Old software was hard to maintain.
   - Now, testing, validation, metrics, and quality standards (ISO, CMMI) are followed to ensure reliability.
8. **Global and Team-Oriented Development**
   - Software development moved from individual programmers to large **teams working across the globe**.

Answer in short:
Software development practices changed from **random coding → structured design → object-oriented methods → agile and iterative processes**, with strong focus on **reuse, quality, maintainability, and teamwork**.

Q1(b). Evolutionary Process Models

Evolutionary models are **iterative software development approaches**. Instead of building the complete system at once, the software is developed **step by step** and improved with user feedback.

Main Evolutionary Models are:

1. **Prototyping Model**
   - A quick prototype (sample system) is developed to understand user needs.
   - Users give feedback, then the prototype is modified until it becomes the final system.
   - **Example:** ATM interface design first shown as prototype before actual coding.

✅ **Advantages:** Helps when requirements are unclear, users see the system early.
❌ **Disadvantages:** May lead to poor design if prototype is not refined properly.

2. **Incremental Model**
   - Software is divided into **increments (parts)**.
   - Each increment adds some features.
   - Users get partial working software early.

- **Example:** A shopping app may first release with login & browsing features, later add cart, payment, delivery tracking.

✅ **Advantages:** Faster delivery, customer satisfaction.
❌ **Disadvantages:** Requires careful planning and design of increments.

3. **Spiral Model**
   - Combines iterative development with **risk analysis**.
   - Each loop has steps: **Planning → Risk Analysis → Engineering → Evaluation**.
   - Used for large and complex projects.

✅ **Advantages:** Best for risky and high-cost projects, reduces failures.
❌ **Disadvantages:** Expensive and complex to manage.

Answer in short:
Evolutionary models build software **gradually with feedback**. Prototyping gives a quick model, Incremental adds features step by step, and Spiral handles risks. These models are flexible and user-friendly compared to old rigid methods like Waterfall.

2a. Explain software development life cycle. Discuss various activities during SDLC.

b. Explain waterfall process model with merits and demerits.

ANSWER:

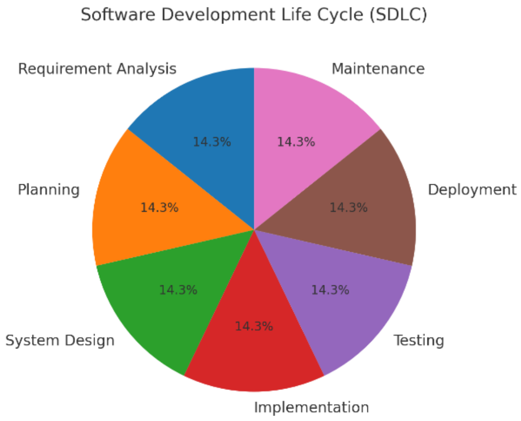
Q2(a). Software Development Life Cycle (SDLC) and Activities

Definition:
Software Development Life Cycle (SDLC) is a **structured process** that describes different stages involved in developing software – from idea to maintenance. It ensures the software is **reliable, efficient, and meets user requirements**.

Activities of SDLC:

1. **Requirement Analysis**
   - Collect and study user needs.
   - Output: **Software Requirement Specification (SRS)** document.
   - Example: Banking system requirements like "withdrawal, deposit, balance check".

2. **Planning**
    - Project plan is prepared: cost, time, manpower, risk analysis.
    - Ensures project is completed on time and within budget.
3. **System Design**
    - Architecture of the system is designed.
    - Includes database, module design, and user interface design.
    - Output: **Design Document**.
4. **Implementation (Coding)**
    - Developers write code using suitable programming languages.
    - Follow coding standards to improve readability and maintainability.
5. **Testing and Validation**
    - Detect and remove errors in the software.
    - Includes **unit testing, integration testing, system testing, acceptance testing**.
    - Ensures software meets requirements.
6. **Deployment (Installation)**
    - Deliver the software to the customer.
    - Installed in real environment for use.
7. **Maintenance**
    - Continuous improvement after release.
    - Fix bugs, update features, improve security.
    - Example: Mobile apps receiving regular updates.



✅ **Summary Answer:**
SDLC covers **Requirement** → **Planning** → **Design** → **Coding** → **Testing** → **Deployment** → **Maintenance**.
It provides a **systematic and disciplined approach** so that the software is **cost-effective, reliable, and user-friendly**.

Q2(b). Waterfall Process Model with Merits and Demerits

Definition:
The **Waterfall Model** is the **earliest process model** in SDLC. It is a **linear sequential model** where each phase is completed before moving to the next. Like a waterfall, the flow is **downwards only**.

Phases of Waterfall Model:

1. **Requirement Analysis** – Gather complete requirements at the beginning.
2. **System Design** – Plan the system's architecture and modules.
3. **Implementation (Coding)** – Convert design into programs.
4. **Testing** – Verify and validate the developed software.
5. **Deployment** – Deliver the software to users.
6. **Maintenance** – Fix errors and enhance features after release.

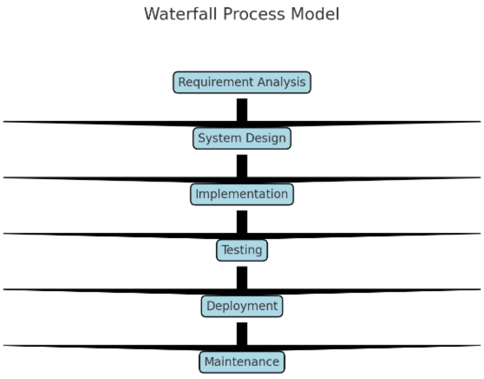
Merits (Advantages):

- **Simple and easy** to understand.
- Well-structured and disciplined.
- Best suited for **small projects** with fixed requirements.

- Easy to manage because each phase has clear deliverables.
- Documentation is strong, which helps in future reference.

Demerits (Disadvantages):

- **Rigid model** – cannot handle requirement changes.
- Customer sees the final product only at the end.
- Testing happens late, so fixing errors is **costly and time-consuming**.
- Not suitable for long, complex, or real-world projects where requirements keep evolving.



✅ **Summary Answer:**
The Waterfall model develops software in a **step-by-step order** (Requirement → Design → Coding → Testing → Deployment → Maintenance).
It is **simple and systematic**, but has **low flexibility** and is not suitable for projects with changing requirements.

3a. Explain the concept of Computer system engineering?

b. Draw and explain the spiral model and list the advantage and disadvantages.

ANSWER:

Q3(a). Explain the Concept of Computer System Engineering

Definition:
Computer System Engineering is a **broader discipline** that deals with the **development of complete computer-based systems**, not just software. It integrates **hardware, software, people, and processes** to build a working system.

Concepts of Computer System Engineering:

1. **Integration of Hardware and Software**
   - Not only software but also hardware components (processors, memory, input-output devices) are considered.
   - Example: ATM machine = software + card reader + cash dispenser + printer.
2. **System as a Whole**
   - Focus is not only on coding but on designing the **entire system** (hardware + software + network + users).
3. **System Constraints**
   - Must satisfy **performance, reliability, cost, and safety requirements**.
4. **Multidisciplinary Nature**
   - Combines knowledge of **electronics, computer engineering, and software engineering**.
5. **Role of Software Engineering**
   - Software engineering is actually a **subset** of computer system engineering.
   - Example: Railway reservation system → hardware servers, networks, terminals + software booking application.

✅ **Answer in short:**
Computer System Engineering is the **umbrella discipline** concerned with developing complete computer-based systems. It includes **hardware + software + people + environment**, while software engineering deals only with the **software part**.

Q3(b). Draw and Explain Spiral Model with Advantages and Disadvantages (10 Marks)

📖 From Pressman (2019) and Rajib Mall (2018)

Definition:
The **Spiral Model** is an **evolutionary software process model** that combines **iterative development** with **systematic risk analysis**. It was proposed by **Barry Boehm**.

Phases of Spiral Model:

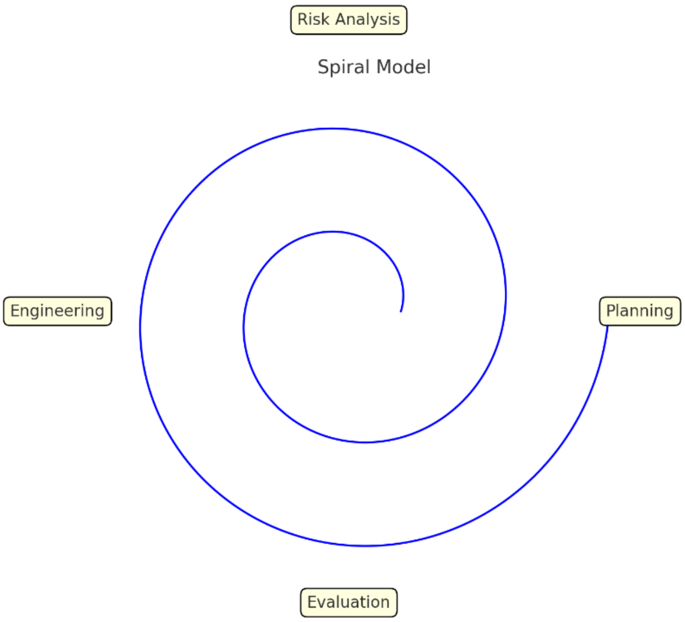
Each spiral loop has **4 main phases**:

1. **Planning** – Identify objectives, requirements, and constraints.
2. **Risk Analysis** – Identify and analyze risks, develop alternatives.
3. **Engineering** – Develop software (design, code, test).
4. **Evaluation** – Customer evaluates the result, feedback is given.

This cycle **repeats for each loop**, with each loop producing a more complete version of the system.

Diagram of Spiral Model:

Let's draw it 👇



Here's the **Spiral Model diagram** you can draw in your exam. It shows the **four phases (Planning, Risk Analysis, Engineering, Evaluation)** arranged in a spiral loop.

Advantages of Spiral Model:

1. Handles **changing requirements** easily.
2. Strong **risk analysis and management**.
3. Customer feedback at every iteration.
4. Useful for **large, complex, and high-risk projects**.
5. Provides working software in early stages.

Disadvantages of Spiral Model:

1. **Expensive** compared to other models.
2. Requires expertise in **risk management**.
3. Difficult to manage for small projects.
4. May take a long time if loops are many.
5. Not suitable if requirements are well understood (Waterfall may be enough).

✅ **Answer in short:**
The Spiral Model develops software in **loops (spirals)**, each with **Planning → Risk Analysis → Engineering → Evaluation**. It is very useful for **large, risky projects** but is **costly and complex** to manage.

4a. Explain about Exploratory style of Developing a software?

b. Explain about Agile Process Models and its principles.

ANSWER:
Q4(a). Exploratory Style of Developing a Software

Definition:
Exploratory style is the **early, unstructured way** of developing software. In this method, programmers directly start coding without proper planning, design, or requirements.

Characteristics:

1. **No clear requirements** – Developers write code as they understand the problem.
2. **No proper design** – Coding begins immediately, no structured design is created.
3. **Trial-and-error development** – Errors are fixed as they appear.
4. **Fast for small tasks** – Works only for very small projects or personal programs.
5. **Not scalable** – As project size increases, this style fails.

Problems in Exploratory Style:

- **No documentation** → difficult to maintain.

- **Poor quality software** due to lack of testing.
- **High cost of maintenance**.
- **Hard to upgrade** when requirements change.
- **Software crisis** occurred in 1960s–70s because many big projects failed due to exploratory style.

✅ **Answer in short:**
Exploratory style means **direct coding without planning, design, or proper testing**. It is only suitable for **tiny programs**, but it led to failures in big projects and gave rise to **modern software engineering practices**.

Q4(b). Agile Process Models and Its Principles

Definition:
Agile process models are **iterative and incremental** development approaches that focus on **flexibility, customer collaboration, and fast delivery**. Agile was introduced as a response to problems in rigid models like Waterfall.

Key Agile Models:

1. **Scrum** – Development in short iterations called **sprints** (2–4 weeks). Daily stand-up meetings, backlog management.
2. **Extreme Programming (XP)** – Emphasizes **continuous testing, pair programming, and customer involvement**.
3. **Crystal, FDD, Lean, Kanban** – Other agile frameworks focusing on simplicity and fast delivery.

Principles of Agile (from Agile Manifesto):

1. **Customer satisfaction** through early and continuous delivery.
2. **Welcome changing requirements** at any stage.
3. Deliver software frequently in **short iterations**.
4. Close **customer–developer collaboration**.
5. Build projects around **motivated individuals**.
6. Face-to-face communication is best.

7. Working software is the **primary measure of progress**.

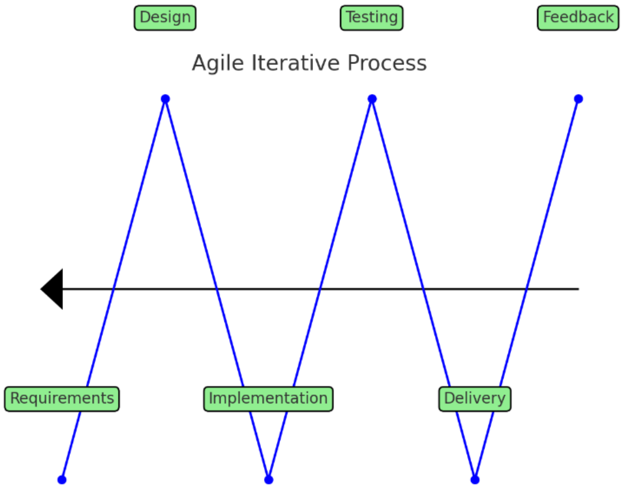
8. Continuous attention to **technical excellence and good design**.

9. Simplicity is essential.

10. Self-organizing teams produce the best results.

11. Regular reflection on how to improve processes.

Diagram of Agile Process (Iteration Cycle):



Here's the **Agile Process Diagram** – it shows an **iterative cycle** (Requirements → Design → Implementation → Testing → Delivery → Feedback → back to Requirements).

Advantages of Agile Models:

- Flexible to **changing requirements**.
- Delivers working software quickly.
- High **customer involvement and satisfaction**.
- Reduces risk of project failure.
- Improves team collaboration.

Disadvantages of Agile Models:

- Hard to estimate cost and time at start.
- Requires highly skilled and motivated teams.
- Not suitable for very large, distributed projects without coordination.
- Continuous customer involvement is necessary.

✅ **Answer in short:**
Agile models are **iterative, flexible, and customer-focused**. They deliver software in **small increments**, welcome changing requirements, and improve quality through continuous collaboration.

5a. Discuss about Software development projects and its types?

b. Perform prototype analysis on “railway reservation” software.

ANSWER:

Q5(a). Software Development Projects and Its Types

Definition:
A **software development project** is an organized activity to create, test, and deliver a software product within a **fixed time, budget, and quality constraints**.

Types of Software Development Projects:

1. **Business Information Systems**
   - Handle business activities like payroll, billing, and accounting.
   - Example: Banking systems, Employee payroll.
2. **Real-Time Systems**
   - Respond to external events within strict timing limits.
   - Example: Air traffic control, Railway signaling.
3. **System Software Projects**
   - Provide platform support for other applications.
   - Example: Operating systems, Compilers, Device drivers.
4. **Engineering and Scientific Applications**
   - Used for simulations, design, and scientific analysis.
   - Example: Weather forecasting, Satellite data processing.
5. **Web-Based Applications**
   - Delivered via the Internet for global access.
   - Example: Railway reservation systems, Online shopping.

6. **Artificial Intelligence (Modern trend)**
    - Use ML/AI for smart decision-making.
    - Example: Chatbots, Recommendation systems, Virtual assistants.

✅ **Summary:**
Software development projects can be of different types such as **business, real-time, system software, scientific, web-based, and AI-based**. Each type has unique **challenges and goals** .

Q5(b). Prototype Analysis on Railway Reservation Software

Definition of Prototype:
A prototype is a **quick working model** of the software used to **collect user feedback** and refine requirements before building the final system.

Steps in Prototype for Railway Reservation System:

1. **Requirement Gathering**
    - Collect basic needs: train search, seat availability, booking, cancellation.
2. **Quick Design**
    - Simple interface screens:
        - Enter source & destination
        - Display trains & availability
        - Booking form
3. **Prototype Building**
    - Build a mock version showing main features like "Search Trains → Book Seats → Ticket Display".
4. **Customer Evaluation**
    - Railway staff and passengers test it.
    - Feedback: add waitlist option, multiple payment modes, seat map.
5. **Refinement**
    - Modify prototype based on feedback.
6. **Final System**

    - Once requirements are clear, the actual robust railway reservation software is developed.

Prototype Model Diagram (Hand-draw in exam):

Requirement Gathering → Quick Design → Prototype Building →

Customer Evaluation → Refinement → Final System

👉 Show feedback arrows looping back from **Customer Evaluation** to **Quick Design** or **Prototype Building**.

Advantages of Prototyping:

- Users understand system early.
- Helps clarify unclear requirements.
- Reduces risk of failure.
- Faster delivery of partial functionality.

Disadvantages of Prototyping:

- May lead to poor design if prototype is not refined.
- Can increase cost if too many iterations.
- Users may think prototype = final system.

✅ **Summary:**
Prototype analysis for the **Railway Reservation System** helps in building a **trial version** first, collecting **feedback**, refining it, and then delivering the final reliable system .

6a. Differentiate between waterfall and incremental model

b. Summarize the Evolution of Software Engineering Methodologies.

ANSWER:

Q6(a). Differentiate between Waterfall and Incremental Model (10 Marks)

Definition of Waterfall Model:

- Linear, sequential approach.
- Each phase must finish before the next starts.

Definition of Incremental Model:

- Software is developed in **increments (small parts)**.
- Each increment adds functionality until full system is complete.

Difference Table (Exam-ready):

Aspect	**Waterfall Model**	**Incremental Model**
Process Flow	Linear and sequential (step by step).	Iterative, divided into increments.
Delivery	Final software delivered only at the end.	Partial working software delivered in each increment.
Flexibility	Rigid, difficult to handle changes.	Flexible, can handle requirement changes easily.
Risk	High risk – errors found late.	Lower risk – errors fixed early in each increment.
Customer Feedback	Customer sees product only at end.	Customer gets feedback after each increment.
Best for	Small projects with fixed requirements.	Medium/large projects where requirements may evolve.
Testing	Done after coding is completed.	Done in every increment (early testing).

✅ **Answer in short:**
Waterfall model is **linear and rigid**, delivering the system at the end, while Incremental model is **iterative and flexible**, delivering software in small working parts with frequent customer feedback .

📌 **Diagram (optional in exam):**

- **Waterfall:** Draw vertical blocks in sequence (Requirement → Design → Coding → Testing → Deployment → Maintenance).
- **Incremental:** Draw stacked bars or loops where each increment delivers part of the system until final system is built.

Q6(b). Summarize the Evolution of Software Engineering Methodologies (10 Marks)

📖 From Unit-1 Notes & Rajib Mall (2018)

Stage 1: Exploratory Style (Before 1960s)

- Programmers directly coded without design or documentation.
- Suitable only for very small programs.
- Led to **software crisis** (delays, failures, high cost).

Stage 2: Early Structured Programming (1960s–1970s)

- Introduction of **structured programming** (control structures, modular coding).
- Systematic documentation and testing started.
- Still weak in handling large complex projects.

Stage 3: Waterfall Model (1970s–1980s)

- Linear, well-defined phases.
- Emphasized planning and documentation.
- Good for stable requirements but rigid.

Stage 4: Evolutionary Models (1980s–1990s)

- Incremental and Prototyping approaches.
- Allowed feedback and flexibility.
- Better suited for dynamic requirements.

Stage 5: Spiral Model (1990s)

- Introduced **risk management**.
- Iterative, with planning → risk analysis → engineering → evaluation.
- Suitable for large, complex projects.

Stage 6: Agile Methodologies (2001 onwards)

- Agile Manifesto introduced.
- Focus on **flexibility, customer collaboration, small iterations, and rapid delivery**.
- Methods include Scrum, XP, Lean, Kanban .

✅ **Answer in short:**
Software engineering methodologies evolved from **Exploratory style → Structured programming → Waterfall → Evolutionary models (Incremental, Prototyping, Spiral) → Agile methods**. This evolution aimed to overcome **software crisis** by improving flexibility, quality, and customer satisfaction .

📌 **Diagram Idea (Timeline):**
Draw a **timeline arrow** with stages:
Exploratory → Structured → Waterfall → Incremental/Prototype/Spiral → Agile

UNIT-2

1a. Explain metrics for project size estimation.

b Discuss the desirable characteristics of a good software requirement specification document.

Q1(a). Metrics for Project Size Estimation

Definition:
Project size estimation means **measuring the size of software** to predict **effort, time, and cost**. Size metrics give a **quantitative measure** of the software to be developed.

Common Metrics for Project Size Estimation:

1. **Lines of Code (LOC):**
   - Count total number of source lines in the program.
   - Simple but language-dependent.
   - Example: 50,000 LOC project → medium-sized.
2. **Function Point (FP):**
   - Measures functionality delivered to the user, not just lines of code.
   - Based on inputs, outputs, inquiries, files, and interfaces.
   - Language-independent.
3. **Use Case Points (UCP):**
   - Size measured using number and complexity of **use cases**.
   - Suitable for object-oriented projects.
4. **Story Points (Agile Metric):**
   - Relative measure of effort for user stories.
   - Used in Agile and Scrum.
   - Example: Story = “Book Railway Ticket” = 5 points.
5. **Other Metrics:**
   - **KDSI/KLOC**: Thousands of Delivered Source Instructions.
   - **Object Points**: Used for GUI, reports, screens.

Diagram (you can draw in exam):

```
Project Size Estimation Metrics
 ├── LOC
 ├── Function Points
 ├── Use Case Points
 ├── Story Points
 └── Object Points
```

✅ **Summary (2–3 lines):**
Project size can be measured by **LOC, FP, UCP, and Story Points**. These metrics help in **effort, cost, and schedule estimation** and are crucial for successful project planning.

Q1(b). Desirable Characteristics of a Good Software Requirement Specification (SRS) Document

Definition:
An **SRS** is a formal document that describes **what the software must do**. A good SRS ensures both developers and customers have a **clear agreement**.

Characteristics of a Good SRS:

1. **Correct**
   - Must correctly capture all customer requirements.
2. **Complete**
   - Covers all features, inputs, outputs, constraints.
   - Nothing important is left out.
3. **Unambiguous**
   - Only one clear meaning, no confusion.
   - Example: Instead of saying "fast response," specify "response time < 2 sec."
4. **Consistent**
   - No contradictions among requirements.
5. **Verifiable**
   - Each requirement should be testable.

   - Example: “System should support 100 users simultaneously.”

6. **Modifiable**
   - Easy to change when requirements evolve.
7. **Traceable**
   - Each requirement linked to its source (customer need/test case).
8. **Prioritized**
   - Shows which requirements are essential, desirable, or optional.
9. **Understandable**
   - Written in simple, structured language (not vague).

Diagram (how to draw in exam):

```
Good SRS Characteristics
 ├── Correct
 ├── Complete
 ├── Unambiguous
 ├── Consistent
 ├── Verifiable
 ├── Modifiable
 ├── Traceable
 └── Prioritized
```

✅ **Summary (2–3 lines):**
A good SRS must be **correct, complete, unambiguous, consistent, verifiable, modifiable, and traceable**. It serves as a **contract** between customer and developer and ensures successful software development.

2a. Describe Software project management complexities in detail.

b. List and explain various requirements elicitation techniques.

ANSWER:

Q2(a). Software Project Management Complexities

Definition:
Software Project Management (SPM) is the discipline of **planning, monitoring, and controlling software projects**. Unlike other engineering projects, software projects face **unique complexities**.

Major Complexities in SPM:

1. **Intangibility of Software**
   - Cannot be seen or touched.
   - Difficult to measure progress and quality.
2. **Changing Requirements**
   - Customers often change needs during development.
   - Adds delays and cost.
3. **Estimation Difficulty**
   - Hard to estimate size, cost, and effort.
   - LOC/FP-based methods may be inaccurate.
4. **Technology Change**
   - Rapidly evolving programming tools, platforms, and hardware.
   - Projects may become outdated quickly.
5. **Human Factor**
   - Software heavily depends on people (skills, communication, teamwork).
   - High attrition increases risk.
6. **Complex Interactions**
   - Multiple modules, teams, and stakeholders need coordination.
   - Miscommunication leads to project failure.
7. **Quality Assurance**
   - Ensuring reliability, security, performance is challenging.

8. **Risk & Uncertainty**
    - Deadlines, budget, and technical risks make projects unpredictable.

Diagram (for exam):

```
Complexities in Software Project Management
 ├── Intangibility
 ├── Changing Requirements
 ├── Estimation Difficulty
 ├── Technology Change
 ├── Human Factor
 ├── Complex Interactions
 ├── Quality Assurance
 └── Risk & Uncertainty
```

✅ **Summary:**
Software project management is complex because of **intangible nature, requirement changes, estimation difficulty, people-dependency, and risks**. This is why systematic project management techniques are essential.

Q2(b). Requirements Elicitation Techniques

Definition:
Requirements elicitation is the process of **gathering requirements** from stakeholders, users, and documents. It is the **first step** in requirements engineering.

Common Techniques:

1. **Interviews**
    - One-to-one or group discussions with stakeholders.
    - Can be structured (fixed questions) or unstructured (open).
2. **Questionnaires/Surveys**
    - Useful when large number of users are involved.
    - Collects opinions quickly but less detailed.

3. **Brainstorming Sessions**
   - Group creativity technique.
   - Helps generate innovative ideas.
4. **Workshops / Focus Groups**
   - Interactive meetings with users & developers.
   - Encourages collaboration.
5. **Observation (Job Shadowing)**
   - Analyst observes users in their working environment.
   - Captures hidden or undocumented requirements.
6. **Document Analysis**
   - Studying existing system documents, manuals, policies.
7. **Prototyping**
   - Build a quick model of the system.
   - Users give feedback, helping refine requirements.
8. **Use Cases & Scenarios**
   - Writing user interactions step by step.
   - Helps clarify functional requirements.

Diagram (for exam):

```
Requirements Elicitation Techniques
  ├── Interviews
  ├── Questionnaires
  ├── Brainstorming
  ├── Workshops
  ├── Observation
  ├── Document Analysis
  ├── Prototyping
  └── Use Cases
```

✅ **Summary:**
Requirements elicitation can be done using **interviews, surveys, brainstorming, observation, document study, prototyping, and use cases**. Using a combination of these ensures **complete and accurate requirements**.

3a. Illustrate the responsibilities of a software project manager in detail.

b. What are the different categories of software development projects according to the COCOMO estimation model? Give an example of software product development projects belonging to each of these categories.

ANSWER:

Q3(a). Responsibilities of a Software Project Manager

Definition:
A **software project manager** is the person responsible for **planning, organizing, directing, and controlling** software projects so that they are delivered **on time, within budget, and with good quality**.

Responsibilities in Detail:

1. **Project Planning**
    - Define scope, objectives, schedules, resources.
    - Prepare project plan (Gantt/PERT charts).
2. **Effort and Cost Estimation**
    - Estimate size using **LOC/FP/COCOMO**.
    - Predict required manpower, time, and budget.
3. **Team Building and Leadership**
    - Select team members, assign roles.
    - Motivate team, solve conflicts.
4. **Scheduling and Tracking**
    - Break project into tasks, set deadlines.
    - Monitor progress against schedule.
5. **Risk Management**

    - Identify risks (technical, financial, human).
    - Prepare mitigation and contingency plans.

6. **Quality Management**
    - Ensure software meets SRS requirements.
    - Apply reviews, testing, and standards.
7. **Communication Management**
    - Ensure smooth communication among stakeholders.
    - Arrange meetings, reports, documentation.
8. **Customer Interaction**
    - Keep clients informed.
    - Manage requirement changes and approvals.
9. **Delivery and Closure**
    - Ensure final product delivery.
    - Post-project review and team feedback.

Diagram (for exam):

```
Responsibilities of Project Manager
 ├── Planning & Estimation
 ├── Team Building
 ├── Scheduling & Tracking
 ├── Risk Management
 ├── Quality Assurance
 ├── Communication
 ├── Customer Interaction
 └── Delivery & Closure
```

✅ **Summary:**
A software project manager is responsible for **planning, estimation, scheduling, risk handling, quality, communication, and delivery**. His role is critical to project success.

Q3(b). Categories of Software Development Projects in COCOMO

Definition:
COCOMO (**Constructive Cost Model**, proposed by Barry Boehm) is an **algorithmic software cost estimation model**. It classifies projects into **three categories** depending on complexity.

COCOMO Categories:

1. **Organic Projects**
    - **Definition:** Small, simple projects; small team; familiar domain.
    - **Characteristics:**
        - Few requirements changes.
        - Experienced team, good communication.
    - **Example:** Payroll system for a small company.
2. **Semi-Detached Projects**
    - **Definition:** Medium size, mixed complexity; team has intermediate experience.
    - **Characteristics:**
        - Some new technology used.
        - Moderate communication needed.
    - **Example:** Inventory control system, University student management system.
3. **Embedded Projects**
    - **Definition:** Large, complex, real-time projects with tight constraints.
    - **Characteristics:**
        - Hardware/software interaction.
        - Very strict requirements (safety, timing).
    - **Example:** Air traffic control software, Railway signaling, Spacecraft software.

Diagram (COCOMO Classification):

COCOMO Project Categories

├── Organic → small, simple, familiar projects

├── Semi-detached → medium size, moderate complexity

└── Embedded → large, complex, real-time projects

✅ **Summary:**
COCOMO classifies projects as **Organic (simple, small)**, **Semi-detached (medium)**, and **Embedded (complex, real-time)**. Each category has **different estimation formulas** for cost and effort.

4a. Briefly explain various project size estimation techniques with an example and compare the advantages and disadvantages of expert judgement and Delphi technique.

b. Explain the differences between functional requirements and nonfunctional requirements by giving suitable examples.

ANSWER:

Q4(a). Project Size Estimation Techniques

Definition:
Project size estimation techniques help us **predict the size of software**, which is used for **effort, cost, and schedule estimation**.

Common Size Estimation Techniques:

1. **Lines of Code (LOC)**
   - Count number of source code lines.
   - Example: A project estimated at 20,000 LOC.
   - *Limitation:* Language dependent.
2. **Function Point (FP)**
   - Measures functionality delivered to user.
   - Based on inputs, outputs, inquiries, files, interfaces.
   - Example: Railway Reservation system → 200 FP.
3. **Use Case Points (UCP)**
   - Based on number and complexity of **use cases**.

- Example: 10 simple, 5 complex use cases → weighted UCP calculated.

4. **Story Points (Agile)**
   - Used in Agile; relative measure of effort for user stories.
   - Example: “Book train ticket” = 5 story points.
5. **Expert Judgement**
   - Experienced managers estimate size based on past projects.
6. **Delphi Technique**
   - Group of experts independently estimate, then results are discussed until agreement is reached.

Comparison of Expert Judgement vs Delphi Technique:

Aspect	**Expert Judgement**	**Delphi Technique**
Who estimates?	One or few experts	Panel of experts
Process	Based on individual past experience	Anonymous independent estimates + group consensus
Bias	May be biased	Reduces bias (group effect)
Accuracy	Depends on single expert’s experience	More reliable (combines opinions)
Time	Quick	Takes more time (several rounds)

✅ **Summary:**
Size estimation techniques include **LOC, FP, UCP, Story Points, Expert Judgement, and Delphi**. Among these, **Delphi is more reliable than single expert judgement** because it uses **consensus**.

Q4(b). Functional vs Nonfunctional Requirements

Definition:

- **Functional Requirements (FR):** Define **what the system should do** (services, inputs, outputs).

- **Non-Functional Requirements (NFR):** Define **how the system should perform** (quality, constraints).

Functional Requirements (FR):

- Describe specific behavior or functions of software.
- Examples:
    1. System must allow users to **search trains by source & destination**.
    2. System must **generate a ticket** after payment.
    3. Admin must be able to **add/update train schedules**.

Non-Functional Requirements (NFR):

- Define quality attributes and constraints.
- Examples:
    1. System should respond **within 2 seconds** (Performance).
    2. System should handle **1000 users simultaneously** (Scalability).
    3. Only authorized users should access (Security).
    4. System should be available **24/7** (Reliability).

Comparison Table:

Aspect	**Functional Requirements**	**Non-Functional Requirements**
Meaning	What the system does	How the system performs
Focus	Services, operations, features	Quality, constraints, performance
Examples	Login, Book ticket, Cancel ticket	Response time, Reliability, Security
Testability	Verified through functional testing	Verified through performance, load, and security testing

---

Diagram (for exam):

```
Requirements
 ├── Functional → Book ticket, Cancel ticket, Search train
 └── Non-Functional → Performance, Security, Usability, Reliability
```

✅ **Summary:**
Functional requirements specify **services/operations**, while non-functional requirements specify **quality and constraints**. Both are essential for a complete and usable system.

5a. What is the goal of requirements analysis phase? Give reasons why the requirements analysis phase is a difficult one.

b. Explain how you can choose the best risk reduction technique when there are many ways of reducing a risk.

ANSWER:

Q5(a). Goal of Requirements Analysis Phase & Why It Is Difficult

Goal of Requirements Analysis Phase:

- To **understand the customer's needs** and **document requirements** clearly before design starts.
- Main objectives:
  1. Define **functional and non-functional requirements**.
  2. Resolve conflicts between stakeholders.
  3. Prioritize requirements.
  4. Prepare **Software Requirement Specification (SRS)** document.

Why Requirements Analysis is Difficult:

1. **Changing Requirements**
   - Customers often change their needs during the project.
2. **Ambiguity in Requirements**
   - Users may describe requirements in vague terms ("system should be fast").
3. **Incompleteness**

   - Customers may not know all requirements at the start.

4. **Conflicting Requirements**
   - Different stakeholders want different features (e.g., user vs. management).
5. **Communication Gap**
   - Misunderstandings between technical team and non-technical customers.
6. **Complexity of System**
   - Large systems (e.g., banking, railway reservation) have too many modules.
7. **Hidden Requirements**
   - Some needs are not expressed but are important (e.g., security rules).

Diagram (for exam):

Goal of Requirements Analysis

├── Identify Requirements

├── Resolve Conflicts

├── Prioritize Needs

└── Prepare SRS

Difficulties

├── Changing Needs

├── Ambiguity

├── Incompleteness

├── Conflicts

├── Communication Gap

└── Complexity

✅ **Summary:**
The goal of requirements analysis is to **clearly define and document software needs**.
It is difficult due to **changing, ambiguous, incomplete, and conflicting requirements**.

Q5(b). Choosing the Best Risk Reduction Technique

Definition of Risk Reduction:
Risk reduction techniques are actions taken to **minimize the probability or impact of risks** in a project. Sometimes, many possible techniques exist, and the manager must choose the best one.

Steps to Choose the Best Risk Reduction Technique:

1. **Identify Alternatives**
    - List all possible techniques (e.g., extra testing, backup servers, training staff).
2. **Analyze Effectiveness**
    - Check how much each technique reduces the risk.
    - Example: For risk of "data loss," alternatives = backup system, cloud storage, RAID disks.
3. **Estimate Cost of Each Technique**
    - Cost includes money, time, manpower.
4. **Cost–Benefit Analysis**
    - Compare **risk reduction achieved vs. cost incurred**.
    - Formula often used:
    - Risk Exposure (RE) = Probability of risk × Loss if risk occurs
    - Risk Reduction Leverage (RRL) = (RE_before – RE_after) / Cost of technique
    - Higher RRL = better technique.
5. **Select Best Technique**
    - Choose the method with **maximum risk reduction leverage**.

Example:

- Risk: System downtime due to server crash.

- Options:
  - Backup server (costly, 90% reduction).
  - Cloud hosting (medium cost, 80% reduction).
  - Extra testing (low cost, 20% reduction).
- Best choice = one with **highest reduction-to-cost ratio** (e.g., cloud hosting).

Diagram (for exam):

```
Risk Reduction Choice
 ├── Identify Alternatives
 ├── Analyze Effectiveness
 ├── Estimate Cost
 ├── Cost-Benefit Analysis
 └── Select Best Technique
```

✅ **Summary:**
When multiple risk reduction techniques are available, the best one is chosen by performing a **cost–benefit analysis** and calculating **risk reduction leverage**. This ensures maximum benefit at minimum cost.

6a. What do you understand by an executable specification language? How is it different from a traditional procedural programming language? Give an example of an executable specification language.

b Discuss the relative advantages of formal and informal requirements specifications.

ANSWER:

Q6(a). Executable Specification Language

Definition:

- An **executable specification language** is a formal specification language that can be **executed (run)** to simulate the behavior of the required system.

- It helps in **validating requirements** by checking how the system would behave even before coding.

Difference from Traditional Procedural Programming Language:

Aspect	**Executable Specification Language**	**Procedural Programming Language**
Purpose	Used to specify and validate requirements	Used to implement actual solution
Abstraction	High-level, describes *what* system should do	Low-level, describes *how* system should do
Execution	Simulates required behavior	Produces final software
Focus	Requirements specification and correctness	Efficiency and performance
User	Analysts, designers, customers	Developers, programmers

Example:

- **Spec#:** a specification language extension of C#.
- **Z notation, VDM, and SDL** are also common formal specification languages.
- These can be executed/simulated to verify requirements.

Diagram (for exam):

```
Executable Specification Language
 ├── High-level requirements model
 ├── Executable for validation
 └── Examples: Z, VDM, Spec#, SDL
```

✅ **Summary:**
An **executable specification language** specifies *what* a system should do and can be executed to simulate system behavior. Unlike procedural programming languages, it is used for **requirement validation, not coding**.

Q6(b). Advantages of Formal vs Informal Requirements Specifications

Definition:

- **Formal specification:** Uses mathematical/logical notations to define requirements.
- **Informal specification:** Uses natural language (English) and diagrams.

Advantages of Formal Specification:

1. **Precision** – No ambiguity, requirements are exact.
2. **Consistency** – Contradictions can be easily detected.
3. **Verifiability** – Easy to prove correctness.
4. **Automation** – Can be analyzed using tools.
5. **Early error detection** – Reduces risk of defects later.

Advantages of Informal Specification:

1. **Simplicity** – Easy to write and understand (natural language).
2. **User-friendly** – Customers without technical background can understand.
3. **Low Cost** – No need for special tools or formal training.
4. **Faster Documentation** – Quick to prepare.
5. **Wide Acceptance** – Suitable for business discussions and contracts.

Comparison Table:

Aspect	**Formal Specification**	**Informal Specification**
Language	Mathematical, logic-based	Natural language, diagrams
Clarity	Very precise, unambiguous	May be vague or ambiguous
Understandability	Hard for non-technical users	Easy for all stakeholders
Verification	Can be validated/proved	Hard to test correctness
Cost	More costly (special skills/tools)	Low cost, fast
Use Case	Safety-critical systems (e.g., aircraft)	General software projects

Diagram (for exam):

Requirement Specifications

├── Formal → precise, verifiable, tool-based

└── Informal → simple, natural language, user-friendly

✅ **Summary:**

- **Formal specifications** are precise, consistent, and verifiable but complex and costly.
- **Informal specifications** are simple, user-friendly, and low-cost but may be ambiguous.

- In practice, many projects use a **combination of both**.

---THE END ---