📘 **UNIT 1: INTRODUCTION**

📌 **Q. Define Software. Explain its Characteristics and Types in Detail. (13–15 marks)**

Definition : Software = Programs + Data + Documentation → set of instructions that tell hardware how to perform tasks → intangible, modifiable, non-wearing.

Characteristics (Detailed):

1. Intangible → Cannot be touched or seen physically → exists as logic and code.
2. Flexible → Easily modified, updated, or extended without physical replacement.
3. Non-wearing → Does not degrade with use → unlike hardware, no physical fatigue.
4. Complex → High interdependency between modules → small change can break entire system.
5. Customizable → Built for specific user needs → can be tailored per client or industry.
6. Evolvable → Can be upgraded over time → new features added without full rebuild.
7. Reusable → Components (libraries, APIs) can be reused across projects → saves cost/time.

Types of Software (With Examples):

1. System Software → Manages hardware/resources → acts as bridge between user + hardware. - Examples: Windows, Linux, Device Drivers. - Function: Memory management, process scheduling.
2. Application Software → Performs user tasks → MS Word, Photoshop, WhatsApp. - Function: Business, education, communication.
3. Embedded Software → Controls devices → ATM, Car ECU, Microwave firmware. - Function: Real-time control, sensor processing.

Real-World Analogy: System SW = Government | App SW = Businesses | Embedded SW = Robots.

Conclusion: Software is the invisible engine of tech — powerful but demands discipline.

📘 **UNIT 2: AGILE SCRUM — 📌 Q. Explain Agile Methodology and Scrum Process in Detail. Include Roles, Phases, Advantages, and Disadvantages. (13–15 marks)**

Agile Manifesto (Core Values): 1. Individuals > Processes 2. Working Software > Documentation 3. Customer Collaboration > Contracts 4. Responding to Change > Following Plan

Scrum Framework: [Product Backlog] → [Sprint Planning] → [Sprint] → [Daily Standup] → [Review] → [Retrospective] → Repeat

Scrum Roles: 1. Product Owner → Defines features, prioritizes backlog. 2. Scrum Master → Removes blockers, enforces process. 3. Dev Team (5–9) → Self-organizing, delivers product.

Scrum Events:

1. Sprint Planning → Set goal, break into tasks.
2. Daily Standup (15 min) → Yesterday? Today? Blockers?
3. Sprint Review → Demo to stakeholders → collect feedback.
4. Sprint Retrospective → What went well? What to improve?

Advantages: - Fast ROI, adaptable, high satisfaction, better morale.
Disadvantages: - Needs experts, scope creep, less docs, not for fixed contracts.
Real-World Use: Startups, Spotify, Amazon.
Conclusion: Scrum turns chaos into controlled iteration — needs discipline and trust.

📘 **UNIT 3: REQUIREMENT ANALYSIS — 📌 Q. Explain Requirement Gathering Techniques and SRS Document in Detail. (13–15 marks)**

Gathering Techniques: 1. Interviews → Deep insights (e.g., bank manager). **2. Surveys** → Scalable feedback (e.g., Google Forms). **3. Observation** → Watch users (e.g., cashier at POS).
4. Prototyping → Mockup → test → refine (e.g., Figma UI).
SRS Purpose: Contract between client + dev → eliminates ambiguity.
SRS Characteristics: Correct, Complete, Consistent, Verifiable, Traceable, Modifiable, Unambiguous.
SRS Structure: 1. Introduction → Purpose, scope.**2. Overall Description** → User needs, constraints.
3. Specific Requirements → Functional + Non-functional.
4. Appendices → Glossary, diagrams.
Pitfalls: Vague terms (“fast”), missing edge cases, ignore performance/security.
Conclusion: Perfect SRS = foundation. Without it = building on quicksand.

📘 **UNIT 4: SYSTEM DESIGN — EXPANDED FOR 13–15 MARK ANSWERS**

📌 **Q. Explain UML Diagrams, DFD, and Design Principles with Examples. (13–15 marks)**

UML Diagrams: 1. Class Diagram → [Student | name,id | enroll()] → structure. **2. Use Case** → (Student) -- (Register) → functionality.

DFD Levels:

- L0: [System] ←→ [User]
- L1: [Login] → [Check Bal] → [Log]
- L2: [Validate PIN] → [Fetch Bal] → [Display]

Design Principles:

1. Functional Independence → High Cohesion + Low Coupling.

- Cohesion: Functional (best), Temporal.
- Coupling: Data (good) → Content (bad).

2. Modularity → Independent modules.

3. Abstraction → Hide complexity → e.g., Car.start().

Conclusion: Good design = understandable, maintainable, scalable — not fancy diagrams.

📘 **UNIT 5: SOFTWARE TESTING — EXPANDED FOR 13–15 MARK ANSWERS**

📌 **Q. Explain Testing Fundamentals, Types, Levels, and Changeover Strategies. (13–15 marks)**

Testing Principles:

1. Early Testing → Fix cheap early.

2. Defect Clustering → 80% bugs in 20% modules.

3. Pesticide Paradox → Update test cases.

4. Shows Presence of Defects → Can't prove defect-free.

5. Absence-of-Errors Fallacy → Bug-free ≠ useful.

Black Box: - Equivalence Partitioning → Valid/Invalid classes (Age 18–60 → test 25, 17, 61). - Boundary Value → Test edges (17,18,59,60,61).

White Box:

- Path/Branch/Statement Coverage → needs code.

Testing Levels:

1. Unit → Individual modules.

2. Integration → Module interactions (Top-down/Bottom-up).

3. System → Full system (Functional, Performance, Security).

4. Acceptance → User validates.

Changeover: 1. Direct → Instant, risky. **2. Parallel** → Old + new, safe, costly. **3. Pilot** → Small group → then full. **4. Phased** → Module by module.

Conclusion: Testing = mindset. Goal = confidence, not perfection.

📘 **UNIT 6: QUESTION TEMPLATES**

🟡 **8–9 MARK TEMPLATE (12–15 lines MAX)**

- Definition (1 line)

- Key Features/Types (3–4 bullets)

- Diagram/Example (3–4 lines)

- Advantages/Disadvantages (2 bullets)

🔴 **13–15 MARK TEMPLATE (20–25 lines MAX)**

- Definition + Purpose (2 lines)

- Diagram-in-Words (5–6 parts)

- Step-by-Step/Phases (4 bullets)

- Advantages + Disadvantages (3 each) - Real-Use + Conclusion (2 lies)