

# 📖 SOFTWARE REQUIREMENTS & ARCHITECTURE (Full Exam Notes)

---

## 1. Software Requirements

### ◆ Definition:

Software requirements are the **needs and conditions** that the software must satisfy to solve a problem or achieve an objective.

### ◆ Types:

1. **Functional Requirements:**
  - What the system **should do**.
  - Example: “System must allow user login.”
2. **Non-functional Requirements:**
  - How the system **should perform**.
  - Example: “System should respond within 2 seconds.”

### ◆ Importance:

- Helps in planning, design, and validation.
- Ensures developers and clients have a **clear understanding**.

✓ **Goal:** Clearly define *what to build* before starting design or coding.

---

## 2. Problem Analysis

### ◆ Definition:

Problem analysis is the process of **understanding the real-world problem**, identifying its **root cause**, and defining **requirements** correctly.

### ◆ Steps in Problem Analysis:

1. **Understand the problem context** – what is the system supposed to solve?
2. **Identify stakeholders** – who will use the system?
3. **Gather information** – through interviews, questionnaires, observation.
4. **Define system boundary** – what’s inside and outside the system.
5. **Identify constraints** – budget, time, technology, etc.

### ◆ Example:

If you're building a "Doctor Appointment System":

- Problem: Patients wait long to book appointments.
- Root cause: Manual booking system.
- Solution: Online appointment booking software.

✓ **Goal:** Understand the *real problem* before defining requirements.

---

## 3. Requirement Specification (SRS)

### ◆ Definition:

A **Software Requirement Specification (SRS)** is a **formal document** that describes what the software will do and how it will perform.

### ◆ Characteristics of a Good SRS (Remember: C-C-U-T-V)

- Clear
- Complete
- Unambiguous
- Testable
- Verifiable

### ◆ Contents of SRS:

1. **Introduction**
  - Purpose, scope, definitions.
2. **Overall Description**
  - Product perspective, functions, constraints.
3. **Specific Requirements**
  - Functional and Non-functional requirements.
4. **External Interface Requirements**
  - Hardware/software interface details.

✓ **Goal:** Provide a **clear contract** between customer and developer.

---

## 4. Functional Specification with Use Cases

### ◆ Definition:

Functional Specification defines **system behavior** (what it should do) in detail.  
Use Cases describe **interactions between user and system**.

### ◆ Use Case Components:

1. **Actors:** Users or external systems.
2. **Use Case:** A goal the actor wants to achieve.
3. **System:** The software that responds to the actor.

### ◆ Example:

Use Case: *Book Appointment*

Element	Description
Actor	Patient
Pre-condition	Patient must be logged in
Main Flow	Select doctor → Choose date → Confirm booking
Post-condition	Appointment is stored in database

✓ **Goal:** Understand how users interact with the system — useful for design and testing.

---

## 5. Validation

### ◆ Definition:

Requirement validation ensures that the **requirements actually reflect the customer's needs** and are correct, complete, and consistent.

### ◆ Methods of Validation:

1. **Reviews:** Peer or stakeholder review of SRS.
2. **Prototyping:** Building a small model to verify user expectations.
3. **Test Case Generation:** Create tests based on requirements to verify correctness.
4. **Walkthroughs:** Step-by-step analysis of requirements.

### ◆ Example:

If the client wanted “appointment reminder via SMS,” and your requirement says “email reminder,” — that’s a validation issue.

✓ **Goal:** Ensure “we are building the *right* product.”

---

## 6. Metrics (for Requirements Phase)

### ◆ Definition:

Metrics are **quantitative measures** used to evaluate the **quality and completeness** of software requirements.

### ◆ Common Requirement Metrics:

Metric	Definition
<b>Completeness</b>	% of identified requirements implemented
<b>Consistency</b>	% of non-conflicting requirements
<b>Correctness</b>	% of requirements that are accurate
<b>Volatility</b>	Number of requirement changes during project

✓ **Goal:** Measure and improve requirement quality and stability.

---

## 7. Role of Software Architecture

### ◆ Definition:

Software architecture defines the **high-level structure** of a software system — how components interact, communicate, and work together.

### ◆ Roles / Importance:

1. **Defines Structure:** Provides blueprint of the system.
2. **Improves Communication:** Common understanding among stakeholders.
3. **Guides Development:** Helps during design and coding.
4. **Ensures Quality Attributes:** Supports performance, scalability, and security.
5. **Enables Reusability:** Reuse architectural patterns in future projects.

### ◆ Example:

In a **web app architecture**, you may have:

- Frontend (React)
- Backend (Node.js)
- Database (MongoDB)
- API connections

✓ **Goal:** Provide a **blueprint** for system development and maintenance.

---

## 8. Architecture Views

### ◆ Definition:

Architecture Views are **different perspectives** used to describe the software architecture to different stakeholders.

### ◆ Common Architecture Views:

View	Description	Example
<b>Logical View</b>	Functional design – what system does	Modules, classes
<b>Process View</b>	Dynamic behavior – how system works	Threads, processes
<b>Development View</b>	Software organization	Packages, files
<b>Physical View</b>	Deployment design	Servers, nodes
<b>Scenario View</b>	How views work together in real use	Use case execution

✓ **Goal:** Explain the system from multiple stakeholder perspectives (developer, tester, client, admin).

---

## 9. Component and Connector (C&C) View

### ◆ Definition:

C&C View represents the **runtime structure** of the system — showing **how components interact via connectors**.

### ◆ Components:

- **Processing elements** like modules, objects, services.  
Example: Authentication Service, Database Module.

### ◆ Connectors:

- **Interaction links** between components (communication, data flow).  
Example: API calls, message queues, HTTP requests.

### ◆ Diagram Example:

```
[User Interface] ---> (API Connector) ---> [Business Logic] ---> (DB Connector) ---> [Database]
```

✓ **Goal:** Show how the software's *parts work together* during execution.

---

## □ QUICK REVISION TABLE

Concept	Meaning	Example	Goal
<b>Software Requirements</b>	What software must do	Login, performance limits	Define system needs
<b>Problem Analysis</b>	Understand problem & root cause	Manual → Online booking	Understand real issue
<b>Requirement Specification (SRS)</b>	Document with all requirements	IEEE 830 format	Formal client-developer agreement
<b>Functional Spec + Use Case</b>	Describe functions & user interactions	Book Appointment use case	Capture user-system behavior
<b>Validation</b>	Check requirements correctness	Review, prototype	Build right product
<b>Metrics</b>	Quantitative quality checks	Completeness, correctness	Measure requirement quality
<b>Role of Architecture</b>	Define system structure	Web app with frontend-backend	Blueprint for development
<b>Architecture Views</b>	Multiple perspectives of system	Logical, Process, Physical views	Communicate with stakeholders
<b>C&amp;C View</b>	Runtime structure – components + connectors	UI ↔ API ↔ DB	Show system interaction

---

## 🏆 EXAM TIPS

- ✓ For **2 marks** → Write definition + one line example.
- ✓ For **5 marks** → Write definition + 3–4 points + example.
- ✓ For **10–15 marks** → Add diagrams + headings + examples + conclusion.
- ✓ Use keywords like “*ensures*,” “*defines*,” “*provides blueprint*,” “*interaction*,” “*validation*” — these score marks easily.