* Requirements Understanding & Lifecycle
* Requirements Gathering Techniques
* Documentation (BRD, FRD, SRS)
* Requirement Validation and Verification
* Traceability Matrix
* Agile Requirements (User Stories, Epics, Backlogs)

Here’s a **comprehensive breakdown**:

**📘 Requirements Engineering & Lifecycle Notes**

**1. Requirements Understanding & Lifecycle**

**Definition**:  
Requirements engineering (RE) is the process of discovering, analyzing, documenting, and managing requirements for a system.

**Lifecycle Phases** (iterative in nature):

1. **Elicitation (Gathering)** → Identify stakeholder needs.
2. **Analysis** → Clarify, refine, and resolve conflicts.
3. **Specification (Documentation)** → Create BRD/FRD/SRS.
4. **Validation & Verification** → Ensure correctness and feasibility.
5. **Management** → Track changes and maintain traceability.

**Example**:  
In a **banking application**:

* Business need = "Enable customers to transfer money online"
* Requirement lifecycle ensures the feature is gathered, analyzed (security, transaction limit), documented, validated (meets compliance), and tracked until delivery.

**2. Requirements Gathering Techniques**

**Techniques** for eliciting requirements from stakeholders:

1. **Interviews** – One-to-one discussion with stakeholders.
   * Example: Asking a branch manager what reports they need from the banking system.
2. **Workshops / JAD (Joint Application Development)** – Collaborative sessions.
   * Example: Business, IT, and compliance sitting together to define KYC process requirements.
3. **Questionnaires / Surveys** – Useful when stakeholders are many.
   * Example: Gathering customer feedback for mobile app features.
4. **Observation / Job Shadowing** – Watching end-users perform tasks.
   * Example: Observing how cashiers handle transactions to design POS software.
5. **Document Analysis** – Studying existing process documents, regulations.
   * Example: Reviewing RBI compliance guidelines for banking apps.
6. **Prototyping** – Creating mockups/wireframes to clarify vague needs.
   * Example: Showing a mobile app UI prototype for funds transfer.

**3. Documentation (BRD, FRD, SRS)**

**a. BRD (Business Requirements Document)**

* High-level business objectives.
* Written from the business/user perspective.
* Focus on **what** the business needs, not **how**.

**Example (Banking App)**:

* "System should allow customers to transfer funds between accounts anytime, anywhere."

**b. FRD (Functional Requirements Document)**

* Translates BRD into detailed system functionality.
* Focus on **how the system will achieve** business needs.

**Example (Fund Transfer)**:

* "System shall validate account number format before transaction."
* "System shall send SMS notification for every successful transfer."

**c. SRS (Software Requirements Specification)**

* A more detailed and technical version (used by developers & testers).
* Includes: functional + non-functional requirements.

**Example (Fund Transfer SRS)**:

* Functional: "System shall allow transfer of max ₹2,00,000 per day."
* Non-functional: "Transaction response time must be < 2 seconds."

**Difference**:

* BRD → Why we need it.
* FRD → What functions it must perform.
* SRS → Detailed technical description for implementation/testing.

**4. Requirement Validation & Verification**

**Validation ("Are we building the right product?")**

* Ensures requirements meet business needs.
* Techniques: reviews, prototypes, stakeholder sign-off.
* Example: Checking with business if the daily transfer limit matches regulatory compliance.

**Verification ("Are we building the product right?")**

* Ensures requirements are correctly implemented.
* Techniques: test cases, walkthroughs, inspections.
* Example: Testing that transfer fails if daily limit is exceeded.

**6. Agile Requirements (User Stories, Epics, Backlogs)**

**User Stories**

* Smallest unit of Agile requirement, written from user perspective.
* Format: *As a [user], I want [function], so that [benefit]*.
* Example:
  + "As a customer, I want to transfer money using UPI so that I can pay instantly."

**Epics**

* Large user requirements broken into multiple stories.
* Example: "Fund Transfer Module" (epic) → includes: UPI transfer, NEFT, IMPS, SMS notifications.

**Backlogs**

* **Product Backlog**: Prioritized list of all features & requirements.
* **Sprint Backlog**: Subset selected for development in current sprint.

**Example (Product Backlog Items)**:

1. Login & Registration
2. Fund Transfer (Epic)
   * Story 1: UPI transfer
   * Story 2: NEFT transfer
   * Story 3: Daily transfer limit
3. Transaction history

**✅ Discussion & Key Takeaways**

* Requirements lifecycle ensures alignment from **business needs → system build → validation/testing**.
* Gathering techniques must be chosen based on stakeholders and project context.
* Documentation clarity (BRD vs FRD vs SRS) prevents misinterpretation.
* Validation ensures business correctness, verification ensures technical correctness.
* Traceability helps avoid "missed" or "untested" requirements.
* Agile approaches simplify requirements into manageable user-centric chunks.

**Traceability Matrix (RTM) – Detailed Notes**

**1. Definition**

A **Traceability Matrix** is a document (usually in Excel or a tool like JIRA, TestRail, HP ALM, etc.) that **maps and traces** user requirements through the project lifecycle.

It ensures:

* Every requirement is **implemented** in design & development.
* Every requirement is **tested and verified**.
* No extra/unnecessary features are built ("scope creep").

**2. Purpose of RTM**

* Ensures **100% test coverage** (no requirement left untested).
* Helps in **impact analysis** (if requirement changes, we know what code/test is affected).
* Acts as an **audit artifact** for compliance (banking, healthcare, aerospace).
* Improves **communication** between business, development, and QA.

**3. Types of Traceability**

1. **Forward Traceability**
   * Maps requirements → design → development → test cases.
   * Ensures that all requirements are covered in implementation & testing.
   * Example: Requirement R1 → Design Doc → Code Module → Test Cases TC01, TC02.
2. **Backward (Reverse) Traceability**
   * Maps test cases → back to requirements.
   * Ensures that we are not testing something unnecessary.
   * Example: TC05 → maps back to R2 (if no mapping, the test is invalid).
3. **Bi-directional Traceability**
   * Combines forward + backward.
   * Most powerful: allows both *requirement-to-test* and *test-to-requirement* mapping.

**4. Structure of a Traceability Matrix**

An RTM is usually a **table**. Common fields:

| **Requirement ID** | **Requirement Description** | **Design Module** | **Code Reference** | **Test Case ID** | **Status** |
| --- | --- | --- | --- | --- | --- |
| R1 | User can transfer funds | TransferService | transfer.java | TC01, TC02 | Pass |
| R2 | OTP authentication | AuthService | auth.java | TC03 | Fail |
| R3 | Daily limit = ₹2,00,000 | LimitService | limit.java | TC04 | Pass |

**5. RTM in SDLC**

* **During Requirements phase** → Each BRD/FRD/SRS requirement gets a unique ID.
* **During Design phase** → Map requirement IDs to design modules.
* **During Development phase** → Link requirement IDs to code components.
* **During Testing phase** → Map requirement IDs to test cases.
* **During UAT/Release** → Verify all requirements are covered and tested.

**6. Example: Banking App RTM**

**Requirement: Fund Transfer**

* **Requirement ID**: R1
* **Description**: "System should allow online fund transfer with OTP authentication."

**Mapping in RTM:**

| **Req ID** | **Requirement** | **Design Module** | **Code Reference** | **Test Case** | **Test Result** |
| --- | --- | --- | --- | --- | --- |
| R1 | Fund transfer with OTP | TransferService, AuthService | transfer.java, otp.java | TC01: Transfer Success  TC02: Invalid OTP | TC01 → Pass  TC02 → Fail |

👉 If TC02 fails, we instantly know requirement R1 is not fully met.

**7. Benefits**

* Reduces risk of **missing requirements**.
* Helps in **regression testing** (when something changes, know which tests to rerun).
* Ensures **regulatory compliance** (RBI, HIPAA, ISO, etc.).
* Provides a **single source of truth** for requirement coverage.

**8. Challenges**

* Time-consuming for **large projects** if done manually.
* Needs regular updates when requirements change.
* Tools (like JIRA, HP ALM, Azure DevOps) are often preferred to maintain RTM dynamically.