**🔹 What is an Epic?**

An **Epic** is a **large body of work** that can be broken down into smaller tasks called **User Stories**. It represents a **high-level goal or feature** and may span multiple sprints.

* Epics are useful for **project planning**, **tracking progress**, and **managing scope**.

**🔹 What is a User Story?**

A **User Story** is a **small, self-contained unit of work** that delivers value to the user. It is written from the **end-user’s perspective** and focuses on **what they want to achieve**.

**📌 User Story Format:**

css

CopyEdit

As a [type of user], I want [an action] so that [a benefit/a goal].

**🔹 Example of an Epic and its User Stories**

**🧩 Epic: User Authentication System**

*Goal*: Allow users to register, login, and manage their sessions securely.

**✅ User Stories under this Epic**

1. **User Story 1: Registration**

pgsql

CopyEdit

As a new user,

I want to sign up using my email and password,

so that I can create an account and access the platform.

* + Acceptance Criteria:
    - Must have email & password fields.
    - Show error for duplicate email.
    - Validate email format.

1. **User Story 2: Login**

pgsql

CopyEdit

As a registered user,

I want to log in using my credentials,

so that I can access my dashboard.

* + Acceptance Criteria:
    - Must support email and password login.
    - Display errors for wrong credentials.
    - Implement “Remember Me” option.

1. **User Story 3: Forgot Password**

pgsql

CopyEdit

As a user who forgot the password,

I want to reset my password via email,

so that I can regain access to my account.

* + Acceptance Criteria:
    - Provide a “Forgot Password” link.
    - Send password reset email.
    - Token expires after 15 minutes.

1. **User Story 4: Logout**

pgsql

CopyEdit

As a logged-in user,

I want to log out securely,

so that my session ends and I’m protected.

* + Acceptance Criteria:
    - Clear session/cookies on logout.
    - Redirect to login screen after logout.

**👥 Roles Involved in Requirement Gathering**

| **Role** | **Responsibility** |
| --- | --- |
| **Product Owner (PO)** | Defines the vision, prioritizes epics/user stories, accepts/rejects deliverables. |
| **Business Analyst (BA)** | Bridges communication between stakeholders and development, gathers detailed requirements. |
| **UI/UX Designer** | Designs user interfaces and experiences based on the requirements. |

**🔁 Flow of Requirement Gathering**

1. **PO** shares the high-level goals (Epics).
2. **BA** refines these into detailed **User Stories**.
3. **UI/UX** creates mockups and wireframes for those stories.
4. All stakeholders review and finalize before development starts.

## 🔍 ****What is Requirement Analysis?****

**Requirement Analysis** is the process of:

* Understanding **what the software needs to do**,
* Identifying **functional** and **non-functional** requirements,
* Converting **business needs** into **technical specifications**.

It serves as the **bridge** between **business requirements** and **system design**.

## 👥 Roles in Requirement Analysis

| **Role** | **Responsibilities** |
| --- | --- |
| **Product Owner (PO)** | Defines the product vision, prioritizes features from the business perspective. |
| **Technical Analyst** | Translates business requirements into technical terms, ensures feasibility. |
| **Software Architect (SW Arch)** | Designs the software architecture — components, interactions, and technology stack. |
| **System Architect (Sys Arch)** | Focuses on system-level architecture — servers, networks, infrastructure, and integrations. |

## 📘 Example: Online Food Delivery System

Let’s walk through **Requirement Analysis → Software Design → Task Breakdown** with an example.

### 🧩 ****Business Requirement**** (From PO):

"We want users to browse restaurants, place food orders, and pay online."

### 🔎 ****Requirement Analysis Output****

#### ✅ Functional Requirements:

1. User Registration/Login
2. Browse restaurants by location
3. Add food items to cart
4. Online payment gateway
5. Order tracking system

#### 🚦 Non-Functional Requirements:

* System should support 10,000 concurrent users.
* Response time should be under 2 seconds.
* High availability: 99.99% uptime.

## 🧠 Involvement of Roles During Analysis

### 🔹 ****Technical Analyst:****

* Checks API integrations (e.g., payment gateway).
* Verifies scalability and data handling (e.g., menu data for 1M+ dishes).

### 🔹 ****Software Architect (SW Arch):****

* Decides on architecture type: Microservices or Monolith?
* Defines data flow between components (user service, order service, payment service).
* Chooses backend tech (e.g., Node.js + MongoDB).

### 🔹 ****System Architect (Sys Arch):****

* Designs infrastructure: cloud provider (AWS), load balancer, auto-scaling.
* Proposes containerization (Docker) and CI/CD pipeline.
* Sets up caching strategies (Redis), database replication.

## 🧱 Software Design (Example Output)

### 🔸 High-Level Components:

| **Module** | **Responsibility** |
| --- | --- |
| **User Service** | Handle login, registration, profile |
| **Restaurant Service** | Store menu, restaurant info |
| **Order Service** | Handle cart, checkout, order status |
| **Payment Service** | Integrate with Stripe/Razorpay |
| **Notification Service** | Send email/SMS for order updates |

### 🔧 Tech Stack Example:

* Backend: Node.js + Express
* Frontend: React
* DB: MongoDB (menu/order), PostgreSQL (user)
* Message Queue: RabbitMQ for async tasks
* DevOps: Docker, Kubernetes, AWS

## 🧩 Task Breakdown Example (For Order Feature)

| **Task** | **Sub-tasks** |
| --- | --- |
| **Order Placement** |  |

* Design order schema (DB)
* API to place order
* Integrate with cart module
* Add validation (e.g., stock, delivery zone)
* Save order to DB
* Queue confirmation email
* Unit and integration tests |

## 🔁 Process Summary

1. **Requirement Gathering** – PO & BA collect business goals
2. **Requirement Analysis** – PO, TA, SW Arch, Sys Arch translate into system specs
3. **Software Design** – Architecture diagrams, DB schema, tech decisions
4. **Task Breakdown** – Divide into developer-friendly, testable pieces

**🧱 What is Task Breakdown?**

Task Breakdown is the process of **decomposing high-level features (user stories)** into **smaller, manageable tasks** that developers can work on. It’s part of sprint planning and ensures that the work is **clear, estimable, and trackable**.

**👥 Roles in Task Breakdown**

| **Role** | **Responsibilities** |
| --- | --- |
| **Software Architect (SW Arch)** | Defines the technical approach and ensures the architecture supports the tasks. |
| **Team Leader (Tech Lead)** | Breaks down features into tasks, estimates effort, assigns responsibilities, reviews feasibility. |
| **Release Manager** | Plans the release cycles, monitors progress across sprints, ensures timely delivery. |

**🔄 Agile Planning Components**

| **Term** | **Description** |
| --- | --- |
| **Sprint** | A time-boxed iteration (usually 1–2 weeks) where a team delivers a set of tasks/user stories. |
| **Iteration** | A cycle of development work, similar to a sprint. |
| **Release Plan** | A roadmap of what features will be delivered in which sprint, and when they’ll be released to users. |

**📘 Example: Feature – “Place an Order” (Online Food Delivery App)**

**🎯 Goal (Epic):**

Allow users to place a food order and pay online.

**✅ User Story:**

As a customer, I want to add food to my cart and place an order, so that I can enjoy a seamless online ordering experience.

**🔨 Task Breakdown (by Team Leader & SW Architect)**

| **Task ID** | **Task** | **Owner** | **Estimation** |
| --- | --- | --- | --- |
| T1 | Design Order schema in DB | Backend Dev | 4 hrs |
| T2 | API to add item to cart | Backend Dev | 6 hrs |
| T3 | API to place order | Backend Dev | 6 hrs |
| T4 | Integrate payment gateway (Stripe) | Backend Dev | 8 hrs |
| T5 | Design "Place Order" UI | Frontend Dev | 6 hrs |
| T6 | Display order confirmation page | Frontend Dev | 4 hrs |
| T7 | Unit & Integration Tests | QA | 8 hrs |
| T8 | Send confirmation email | Backend Dev | 4 hrs |

**🗓️ Sprint Planning**

| **Sprint** | **Tasks Planned** | **Duration** |
| --- | --- | --- |
| Sprint 1 | T1, T2, T5, T6 | 1 week |
| Sprint 2 | T3, T4, T8 | 1 week |
| Sprint 3 | T7, Bug Fixes, Review | 1 week |

**📦 Release Plan**

| **Version** | **Features** | **Release Date** |
| --- | --- | --- |
| v1.0 | Register, Browse Menu, Place Order (without payment) | May 15 |
| v1.1 | Online Payment + Order Tracking | May 30 |
| v1.2 | Promo Codes + Scheduled Orders | June 15 |

🔁 **Release Manager** coordinates this, ensures testing & deployment are aligned with the schedule.

**🔄 Flow Summary:**

1. **User Story selected**
2. **SW Architect** sets design guidelines
3. **Team Leader** breaks down into dev/test/design tasks
4. **Effort estimation** and **assignment**
5. **Release Manager** aligns these tasks to a **sprint** and **release milestone**

**🔧 What is the Development Phase?**

The **Development Phase** is where the **actual coding** happens, based on the **task breakdown** and **design decisions** made earlier. It involves writing code, testing, reviewing, and preparing for integration and deployment.

**👥 Roles in the Development Phase**

| **Role** | **Responsibilities** |
| --- | --- |
| **Developers** | Implement features, write unit tests, fix bugs. |
| **Team Leader (Tech Lead)** | Guides developers, reviews architecture alignment, ensures code standards. |
| **Software Architect (SW Arch)** | Ensures scalability, security, modularity in implementation. |
| **Site Reliability Engineer (SRE)** | Sets up monitoring/logging, ensures high availability and deployment readiness. |

**🔄 Activities in Development**

| **Activity** | **Description** |
| --- | --- |
| **Feature Implementation** | Developers write code for user stories. |
| **Unit Testing** | Each component is tested independently using automated tests. |
| **Peer Review / Pair Programming** | Code is reviewed by team members or written together in pairs. |
| **Code Quality Control** | Enforced via linters, static code analysis tools, and review checklists. |

**📘 Example: Implement “Add to Cart” Feature (E-Commerce App)**

**🔹 Step 1: Feature Implementation**

**Developer writes code:**

python

CopyEdit

# cart.py

def add\_to\_cart(user\_id, product\_id, quantity):

cart = db.get\_cart(user\_id)

cart.add\_item(product\_id, quantity)

db.save(cart)

**🔹 Step 2: Unit Testing**

**Developer writes tests:**

python

CopyEdit

# test\_cart.py

def test\_add\_to\_cart\_success():

user\_id = 1

product\_id = 101

result = add\_to\_cart(user\_id, product\_id, 2)

assert result['status'] == 'success'

* Tools used: pytest, unittest, Jest (for JS), etc.

**🔹 Step 3: Peer Review / Pair Programming**

**Code is reviewed:**

* Reviewer checks:
  + Code readability
  + Naming conventions
  + Edge case handling
  + Reusability
  + Security issues (e.g., SQL injection)

Tools used: GitHub PRs, GitLab Merge Requests, Bitbucket, etc.

**🔹 Step 4: Code Quality Control**

**Automated tools:**

* **Static analysis**: SonarQube, ESLint, Pylint
* **Code style**: Prettier, Black
* **CI pipeline**: Run unit tests, check coverage

**🔐 SRE Responsibilities During Development**

| **Task** | **Description** |
| --- | --- |
| Set up logging | Configure log aggregators (e.g., ELK, Grafana) |
| Monitoring | Use Prometheus, Datadog to track API performance |
| Error tracking | Sentry, New Relic integration |
| Deployment scripts | Dockerfiles, Kubernetes manifests, CI/CD setup |

**✅ Development Workflow Summary**

1. **Developer picks task → implements code**
2. **Writes unit tests → runs tests locally**
3. **Sends Pull Request → peer reviews it**
4. **Team Lead approves and merges code**
5. **SRE sets up CI/CD, logs, monitors reliability**
6. **SW Architect ensures design standards are followed**

**🔁 Development in a DevOps Environment**

In a **DevOps culture**, development and operations collaborate closely to:

* Automate testing & deployment
* Ensure continuous integration (CI)
* Deliver features quickly and reliably

**👥 Roles and Responsibilities**

| **Role** | **Responsibilities** |
| --- | --- |
| **Developers** | Implement features, write unit/integration tests, push code to repo |
| **Team Leader (Tech Lead)** | Reviews code, enforces standards, ensures test coverage |
| **DevOps Engineer** | Automates testing and deployment pipelines, configures CI/CD, monitors code quality |

**🧩 Key Concepts**

**✅ 1. Unit Testing**

* **What**: Testing individual functions or classes in isolation.
* **Who**: Developers write and run them.
* **Tools**: JUnit (Java), Pytest (Python), Jest (JavaScript), NUnit (C#)

**Example (Python)**:

python

CopyEdit

def add(x, y):

return x + y

def test\_add():

assert add(3, 2) == 5

**✅ 2. Automated Unit Testing**

* Unit tests are automatically run by CI tools whenever code is pushed.
* Ensures every small unit of code works correctly.
* Can block pull requests if any test fails.

**Tools**: GitHub Actions, GitLab CI, Jenkins, CircleCI

**✅ 3. Automated Integration Testing**

* **What**: Tests how multiple components/services work together.
* **Who**: Developers + QA
* **When**: After unit tests pass
* **Example**: Testing login API + database + session service together

**Automation**:

* Triggered in CI after build stage
* Often run in a containerized test environment (Docker)

**✅ 4. Peer Review**

* **What**: Developers review each other's code
* **Why**:
  + Catch bugs early
  + Improve readability
  + Share knowledge
* **Tools**: GitHub PRs, GitLab Merge Requests, Bitbucket

**Checklist**:

* ✅ Clean and readable code
* ✅ Proper variable/method naming
* ✅ Tests written and passing
* ✅ No sensitive data or hardcoded credentials

**✅ 5. Automatic Code Quality Checks**

* **What**: Static code analysis tools analyze code for:
  + Complexity
  + Security flaws
  + Duplicates
  + Code style violations
* **When**: Runs in CI/CD pipeline automatically
* **Who Monitors**: DevOps and Team Lead

**Tools**:

* **SonarQube** – code quality & security
* **ESLint/TSLint** – JS/TS style checkers
* **Pylint/Black** – Python
* **Checkstyle/PMD** – Java

**Example Output**:

text

CopyEdit

[SonarQube] 3 bugs, 5 code smells, 2 security vulnerabilities detected.

Build blocked until fixed.

**🔁 Typical CI/CD Pipeline (with DevOps Involvement)**

plaintext

CopyEdit

1. Developer pushes code ⟶

2. Unit Tests (automated) run ⟶

3. Integration Tests run ⟶

4. Code Quality Check (SonarQube, linters) ⟶

5. Peer Review by teammates ⟶

6. Merge to main branch ⟶

7. Deployment to staging/production

**✅ Summary Table**

| **Phase** | **Responsibility** | **Automation** |
| --- | --- | --- |
| **Unit Testing** | Developer | ✅ GitHub Actions, Jenkins |
| **Integration Testing** | DevOps + QA | ✅ Docker/Test Environment |
| **Peer Review** | Developers/Team Lead | 🚫 Manual but enforced in process |
| **Code Quality Check** | DevOps + Team Lead | ✅ SonarQube, Linters |

**🚀 What is SIT (System Integration Testing)?**

**SIT** is a **staging-like environment** where the **integrated system** is tested as a whole before moving to UAT or production.  
It focuses on **integration points**, **workflow correctness**, and **end-to-end scenarios**.

**👥 Roles Involved**

| **Role** | **Responsibilities** |
| --- | --- |
| **DevOps Engineer** | Deploys the build to the SIT environment, sets up test automation pipelines, ensures environment is production-like |
| **Team Leader (Tech Lead)** | Coordinates deployment, assigns bug fixes, validates build stability |
| **Developers** | Fix integration bugs, assist in debugging, validate feature functionality |

**🛠️ Deployment to SIT – What Happens?**

1. Code from main or release branch is packaged.
2. DevOps automates the **deployment using CI/CD** tools (e.g., Jenkins, GitHub Actions, GitLab CI).
3. The system is **deployed to the SIT environment** (cloud or on-prem).
4. **Tests (automated/manual)** are triggered post-deployment.
5. The **Team Lead** and **QA team** verify if the build is stable for further testing.

**🧪 Types of Testing in SIT**

| **Test Type** | **Description** | **Automated / Manual** |
| --- | --- | --- |
| **Smoke Testing** | A basic set of tests to verify that major functionalities are working (e.g., login, dashboard load, APIs respond) | ✅ Can be automated |
| **End-to-End (E2E) Testing** | Simulates real-world user scenarios from start to finish (e.g., “Login → Browse → Add to Cart → Checkout”) | ✅ Automated or 🧑 Manual |
| **Regression Testing** | Re-runs previous test cases to ensure new code hasn’t broken existing features | ✅ Automated |

**⚙️ Example Workflow: E-Commerce Checkout**

**Feature: Checkout Flow**

“As a user, I want to purchase items and receive an order confirmation.”

**✅ Smoke Test Scenarios:**

* Can log in successfully?
* Can access checkout page?
* Is payment gateway reachable?
* Can log out?

**✅ E2E Test Scenario:**

plaintext

CopyEdit

1. Open website

2. Login

3. Add items to cart

4. Go to checkout

5. Enter shipping address

6. Pay using credit card

7. See confirmation message

**Automation Tools**:

* **Cypress** (JavaScript-based, popular for frontend E2E testing)
* **Selenium WebDriver**
* **Playwright** (for modern web apps)

**🔄 Automated vs Manual Testing in SIT**

| **Type** | **Best For** | **Tools** | **Trigger** |
| --- | --- | --- | --- |
| **Automated** | Repetitive, regression, smoke tests | Cypress, Selenium, Postman/Newman, Jenkins | CI/CD pipeline |
| **Manual** | UI checks, exploratory testing, edge cases | QA team, Browser Dev Tools | After deployment |

**📦 Deployment Flow Example (CI/CD)**

plaintext

CopyEdit

1. Developer pushes code ⟶

2. Build & Unit Tests run (CI) ⟶

3. DevOps deploys to SIT ⟶

4. Run Automated Smoke Tests ⟶

5. Run Automated E2E Tests ⟶

6. Manual Exploratory Testing by QA/Developers ⟶

7. If stable, promote to UAT

**✅ Summary Table**

| **Stage** | **Activity** | **Who** | **Automation** |
| --- | --- | --- | --- |
| Deploy to SIT | Package & deploy release | DevOps | ✅ Jenkins/GitHub Actions |
| Smoke Testing | Quick checks of system health | QA / Dev | ✅ (preferred) |
| End-to-End Testing | Full user journeys | QA / Dev | ✅ + 🧑 manual |
| Bug Fixes | Patch issues found | Developers | 🚫 (manual) |
| Build Validation | Sign-off for UAT | Team Lead | 🚫 (manual) |