🔹 **Understanding an Epic**  
An **Epic** is a substantial body of work that can be decomposed into smaller, deliverable items known as User Stories. It serves as a high-level representation of a major feature or objective and may span several sprints.  
• **Benefits:** Epics assist with strategic planning, progress tracking, and scope management.

🔹 **Defining a User Story**  
A **User Story** is a concise, standalone unit of work designed to deliver specific value to the end-user. It is articulated from the user's perspective and focuses on their needs and benefits.  
📌 **User Story Template:**

css

CopyEdit

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

🔹 **Illustrative Epic and Its User Stories**  
🧩 **Epic: User Authentication System**  
**Objective:** Implement secure registration, login, and session management for users.

✅ **User Stories Under This Epic**

1. **User Story 1: Registration**

pgsql

CopyEdit

As a new user,

I want to sign up with my email and password,

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

**Acceptance Criteria:**

* + Include email and password fields.
  + Provide an error message for duplicate email entries.
  + Verify the email format is correct.

1. **User Story 2: Login**

pgsql

CopyEdit

As a registered user,

I want to log in with my credentials,

so that I can access my personal dashboard.

**Acceptance Criteria:**

* + Support email and password authentication.
  + Display error messages for invalid credentials.
  + Implement a “Remember Me” functionality.

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

pgsql

CopyEdit

As a user who has forgotten my password,

I want to reset it via an email link,

so that I can regain access to my account.

**Acceptance Criteria:**

* + Provide a "Forgot Password" option.
  + Issue a password reset email.
  + Ensure the reset token expires in 15 minutes.

1. **User Story 4: Logout**

pgsql

CopyEdit

As an authenticated user,

I want to log out securely,

so that my session is safely terminated.

**Acceptance Criteria:**

* + Clear all session data or cookies upon logout.
  + Redirect the user to the login screen after logout.

👥 **Key Roles in Requirement Collection**

| **Role** | **Responsibilities** |
| --- | --- |
| **Product Owner (PO)** | Articulates the vision, sets priorities for epics/user stories, and approves or rejects deliverables. |
| **Business Analyst (BA)** | Acts as a mediator between stakeholders and the development team, gathering detailed requirements. |
| **UI/UX Designer** | Develops user interface designs and experiences based on the requirements. |

🔁 **Flow of Requirement Gathering**

1. The PO presents high-level objectives (Epics).
2. The BA refines these into clear, detailed User Stories.
3. The UI/UX Designer creates wireframes and mockups based on these stories.
4. Stakeholders collaborate to review and finalize the requirements before starting development.

🔍 **What is Requirement Analysis?**  
Requirement Analysis involves:  
• Comprehending the functional needs of the software,  
• Identifying both functional and non-functional requirements, and  
• Translating business needs into technical specifications.  
This process connects business objectives with system design.

👥 **Roles in Requirement Analysis**

| **Role** | **Responsibilities** |
| --- | --- |
| **Product Owner (PO)** | Provides the overall product vision and prioritizes features from the business standpoint. |
| **Technical Analyst (TA)** | Converts business requirements into technical details and examines project feasibility. |
| **Software Architect (SW Arch)** | Designs the application architecture including component interactions and selecting the tech stack. |
| **System Architect (Sys Arch)** | Concentrates on the system’s infrastructure such as servers, networks, integrations, and scalability. |

📘 **Case Study: Online Food Delivery Application**  
**Business Objective (From PO):**  
"We need an application where users can browse restaurants, order food, and complete online payments."

🔎 **Outcome of Requirement Analysis**

✅ **Functional Requirements:**

1. User Registration & Login
2. Browsing restaurants based on location
3. Adding food items to a shopping cart
4. Integration with an online payment gateway
5. Order tracking capability

🚦 **Non-Functional Requirements:** • The system should handle 10,000 simultaneous users.  
• Response time must be below 2 seconds.  
• Uptime should be maintained at 99.99%.

🧠 **Participation During Analysis**

**Technical Analyst:**  
• Evaluates API integrations, especially for payment gateways.  
• Assesses scalability and data management, e.g., handling millions of dish entries.

**Software Architect (SW Arch):**  
• Chooses the type of architecture (Microservices vs. Monolithic).  
• Defines data flows between services like user, order, and payment services.  
• Selects appropriate backend technologies (e.g., Node.js with MongoDB).

**System Architect (Sys Arch):**  
• Outlines the infrastructure including cloud providers, load balancing, and auto-scaling strategies.  
• Proposes the use of Docker containers and CI/CD pipelines.  
• Determines caching strategies and database replication setups.

🧱 **Software Design (Example Outcome)**

🔸 **Core Components:**

| **Module** | **Responsibility** |
| --- | --- |
| **User Service** | Manages user registration, login, and profile. |
| **Restaurant Service** | Contains restaurant details and menu information. |
| **Order Service** | Facilitates cart management, checkout, and order tracking. |
| **Payment Service** | Integrates payment solutions (e.g., Stripe, Razorpay). |
| **Notification Service** | Sends updates via email or SMS regarding order status. |

🔧 **Example Tech Stack:**

* **Backend:** Node.js with Express
* **Frontend:** React
* **Databases:** MongoDB for menus/orders and PostgreSQL for user details
* **Message Queue:** RabbitMQ for asynchronous processing
* **DevOps:** Containerization with Docker, orchestration with Kubernetes, and cloud services such as AWS

🧩 **Task Breakdown Example (for Order Placement Feature)**

| **Task** | **Sub-Tasks** |
| --- | --- |
| **Order Placement** | - Design the database schema for orders |

* Develop the API to initiate an order
* Integrate seamlessly with the shopping cart
* Validate stock levels and delivery constraints
* Store the order in the database
* Queue a confirmation email
* Write unit and integration tests |

🔁 **Process Recap**

1. **Requirement Gathering:** PO & BA collect business objectives.
2. **Requirement Analysis:** PO, TA, SW Arch, and Sys Arch convert these into technical specifications.
3. **Software Design:** Creation of architectural diagrams, DB schemas, and tech selection.
4. **Task Breakdown:** Decomposing work into small, testable tasks for developers.

🔹 **Understanding Task Breakdown**  
Task Breakdown involves splitting high-level requirements (User Stories) into smaller, easily managed tasks that are assigned during sprint planning. This ensures each task is clear and can be estimated effectively.

👥 **Key Roles in Task Breakdown**

| **Role** | **Responsibilities** |
| --- | --- |
| **Software Architect (SW Arch)** | Defines technical approaches ensuring the design supports individual tasks. |
| **Team Leader (Tech Lead)** | Decomposes features into tasks, estimates time/effort, and allocates work. |
| **Release Manager** | Coordinates sprints, monitors progress, and aligns tasks with release milestones. |

🔄 **Agile Planning Components**

* **Sprint:** A fixed time-box (usually 1–2 weeks) where specific tasks/User Stories are completed.
* **Iteration:** Cycles of development work, similar to sprints.
* **Release Plan:** An overall roadmap outlining which features will be delivered in which sprints, and the planned release dates.

📘 **Example: Feature – "Place an Order" (For an Online Food Delivery App)**  
**Epic (High-Level Objective):**  
Enable users to order food and pay online.

✅ **User Story:**

pgsql

CopyEdit

As a customer, I want to add items to my cart and place an order, so that I can easily complete my purchase.

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

| **Task ID** | **Task Description** | **Assigned To** | **Estimated Time** |
| --- | --- | --- | --- |
| T1 | Design the database schema for orders | Backend Dev | 4 hours |
| T2 | Develop the API to add items to the cart | Backend Dev | 6 hours |
| T3 | Create the API for order placement | Backend Dev | 6 hours |
| T4 | Integrate payment via Stripe | Backend Dev | 8 hours |
| T5 | Design the UI for "Place Order" | Frontend Dev | 6 hours |
| T6 | Build the order confirmation screen | Frontend Dev | 4 hours |
| T7 | Develop unit & integration tests | QA | 8 hours |
| T8 | Implement confirmation email system | Backend Dev | 4 hours |

🗓️ **Sprint Planning Example**

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

📦 **Release Plan Example**

| **Version** | **Features Included** | **Planned Release Date** |
| --- | --- | --- |
| v1.0 | User Registration, Menu Browsing, Order Placement (excluding payment) | May 15 |
| v1.1 | Online Payment Integration and Order Tracking | May 30 |
| v1.2 | Promo Codes and Scheduled Orders | June 15 |

The **Release Manager** ensures everything is aligned with this schedule, coordinating with testing and deployment processes.

🔄 **Summary of the Development Phase**  
This phase is where actual coding occurs based on the breakdown of tasks and design decisions. It includes code development, testing, reviews, and preparing for integration and deployment.

👥 **Roles in Development**

| **Role** | **Responsibilities** |
| --- | --- |
| **Developers** | Write and test code, fix bugs, implement features. |
| **Team Leader (Tech Lead)** | Assist developers, review code for architecture and standards. |
| **Software Architect (SW Arch)** | Maintain overall design integrity, scalability, and security. |
| **Site Reliability Engineer (SRE)** | Set up monitoring/logging, and ensure system stability in deployment. |

🔄 **Key Activities in Development**

* **Feature Implementation:** Writing the code based on the user story.
* **Unit Testing:** Testing individual functions or components using automated tests.
* **Peer Review / Pair Programming:** Collaborating with team members to review code for clarity and adherence to best practices.
* **Code Quality Control:** Enforcing standards via linters, static analysis tools, and code review checklists.

🔐 **DevOps & SRE Involvement During Development**

* **Logging Setup:** Configure logging systems (e.g., ELK stack, Grafana).
* **Monitoring:** Use tools like Prometheus or Datadog to monitor performance.
* **Error Tracking:** Integrate tools like Sentry or New Relic.
* **Deployment Automation:** Develop Dockerfiles, Kubernetes manifests, and CI/CD pipelines.

✅ **Summary Workflow in Development**

1. **Task Assignment:** Developer selects a task and writes code.
2. **Testing:** Implement unit tests and run them locally.
3. **Peer Review:** Submit a pull request for code review.
4. **Approval & Merge:** Team Lead approves and merges the code after review.
5. **Deployment:** SRE ensures CI/CD processes are in place, including logging and monitoring.
6. **Architectural Verification:** SW Architect verifies that design standards are maintained throughout the development.

🔄 **Development in a DevOps Environment**

Agile teams working in a DevOps culture collaborate closely to:  
• Automate the testing and deployment process,  
• Ensure continuous integration (CI) for every commit,  
• Rely on continuous delivery to deploy features reliably.

👥 **Roles & Responsibilities in a DevOps Setup**

| **Role** | **Responsibilities** |
| --- | --- |
| **Developers** | Write code, develop unit/integration tests, and push changes. |
| **Team Leader** | Enforce coding standards, review pull requests, and ensure test coverage. |
| **DevOps Engineer** | Automate pipelines, configure CI/CD, and monitor deployments. |

🧩 **Key Concepts in Testing & Code Quality**

1. **Unit Testing:**
   * **Purpose:** Validate individual code units in isolation.
   * **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

1. **Automated Unit Testing:**
   * **Practice:** Run unit tests automatically using CI tools.
   * **Purpose:** Ensure code reliability at every commit and prevent regressions.
2. **Integration Testing:**
   * **Purpose:** Verify interactions between different system components.
   * **Approach:** Typically automated or performed manually if necessary, especially for end-to-end workflows.
3. **Peer Review:**
   * **Objective:** Enhance code quality through collaborative review.
   * **Benefits:** Early bug detection, improved code readability, and increased team knowledge sharing.
   * **Tools:** GitHub Pull Requests, GitLab Merge Requests, Bitbucket.
4. **Automated Code Quality Checks:**
   * **Tools:** SonarQube for security and quality, ESLint/TSLint for JavaScript/TypeScript, Pylint/Black for Python, Checkstyle/PMD for Java.
   * **Outcome:** Reports issues related to code complexity, security vulnerabilities, and style inconsistencies, blocking builds until resolved.

🔄 **CI/CD Pipeline (Typical Flow in DevOps)**

1. **Developer** pushes code →
2. **Unit Tests** run via CI tools →
3. **Integration Tests** execute →
4. **Code Quality Tools** evaluate the code (e.g., SonarQube, linters) →
5. **Peer Reviews** are conducted →
6. Code is merged to the main branch →
7. Deployment to staging/production environments follows.

✅ **System Integration Testing (SIT) Overview**  
SIT provides a stage-like environment where the integrated software is tested as a whole before moving to User Acceptance Testing (UAT) or production. The focus is on verifying all integration points, validating workflows, and ensuring end-to-end system functionality.

👥 **SIT Key Roles**

| **Role** | **Responsibilities** |
| --- | --- |
| **DevOps Engineer** | Manages the deployment of builds to the SIT environment and configures test automation pipelines. |
| **Team Leader (Tech Lead)** | Oversees the deployment process, coordinates bug fixing, and confirms build stability. |
| **Developers** | Fix integration issues and support debugging of complex scenarios. |

🛠️ **Deployment Process in SIT**

1. The build from the main/release branch is packaged.
2. The DevOps team deploys the build to a SIT environment using CI/CD tools (e.g., Jenkins, GitHub Actions).
3. Automated and/or manual tests run post-deployment.
4. The Team Lead and QA confirm that the build is stable and ready for further testing.

🧪 **Testing Types in SIT**

| **Test Type** | **Description** | **Automation** |
| --- | --- | --- |
| **Smoke Testing** | Basic tests to verify that critical functionalities are active (e.g., login, dashboard, APIs) | Typically automated |
| **End-to-End Testing** | Simulates complete user scenarios from beginning to end (e.g., Login → Browse → Add to Cart → Checkout) | Can be both automated and manual |
| **Regression Testing** | Re-executes earlier test scenarios to confirm that new changes have not broken any features | Usually automated |

⚙️ **Example Flow: E-Commerce Checkout Process**

**Feature:** Checkout Process  
*User Story:* “As a user, I want to complete a purchase and receive an order confirmation.”

**Smoke Test Scenarios:**

* User can log in successfully.
* Checkout page is accessible.
* Payment gateway is responsive.
* User can log out securely.

**End-to-End Test Scenario:**

1. Open the website.
2. Log in.
3. Add items to the shopping cart.
4. Proceed to checkout.
5. Enter a shipping address.
6. Complete the payment using a credit card.
7. Receive an order confirmation message.

**Tools for Automation:**

* Cypress (for frontend E2E tests)
* Selenium WebDriver
* Playwright (modern web apps)

🔄 **Automated vs. Manual Testing in SIT**

| **Testing Type** | **Best Suited For** | **Tools** | **Trigger Mechanism** |
| --- | --- | --- | --- |
| **Automated** | Repeated, regression, and smoke tests | Cypress, Selenium, Postman/Newman, Jenkins | CI/CD pipeline |
| **Manual** | UI/UX checks, exploratory testing, and complex edge cases | QA teams with browser developer tools | Post-deployment testing |

📦 **Example Deployment Pipeline via CI/CD**

1. **Developer** commits code →
2. **CI** initiates build and runs unit tests →
3. **DevOps** deploys the build to the SIT environment →
4. **Automated Smoke Tests** are executed →
5. **Automated End-to-End Tests** run →
6. **Manual Exploratory Testing** is conducted by the QA team →
7. Upon confirmation of stability, the build is promoted to UAT.

✅ **Final Summary Table**

| **Phase/Stage** | **Responsibility** | **Automation Tool/Approach** |
| --- | --- | --- |
| **Unit Testing** | Developer-written tests | GitHub Actions, Jenkins |
| **Integration Testing** | Coordination between DevOps and QA teams | Docker/Test Environment, Automated Scripts |
| **Peer Review** | Code reviews and collaborative assessments | GitHub PRs, GitLab MRs |
| **Code Quality Checks** | Static analysis and linters | SonarQube, ESLint, Pylint |