

Contents

Software Feature Development Cycle Overview	1
Feature Lifecycle	1
Task Lifecycle	2
Feature Definition	3
Key Stakeholders Involved	3
Core Questions and Requirements	3
Approval	4
Feature Testing & Deployment	4
Test Deployment (QA Release)	4
Feature Testing	5
Defect Remediation and Approval	5
Deployment Approval	5
Feature Deployment (Production)	5
Task Integration	6
Task Deployment to Dev/Integration	6
Task Testing in QA Environment	6
Defect Handling	6

Software Feature Development Cycle Overview

This document outlines the end-to-end feature-based development process using a Scrum-like methodology, focusing on structured definition, implementation, and deployment phases.

Feature Lifecycle

The Feature Cycle is the top-level process that tracks a user-facing capability from conception to production release.

Stage	Description
Feature Definition	Ideation, scope, and requirements are defined and approved. more. . .
Task Definition	The feature is broken down into small, implementable development tasks.
Task Cycle	Tasks are implemented, deployed to a development environment, and tested.
Feature Testing	The completed feature is tested end-to-end in a quality assurance (QA) environment.

Stage	Description
Feature Deployment	The feature is approved and deployed to the production environment.

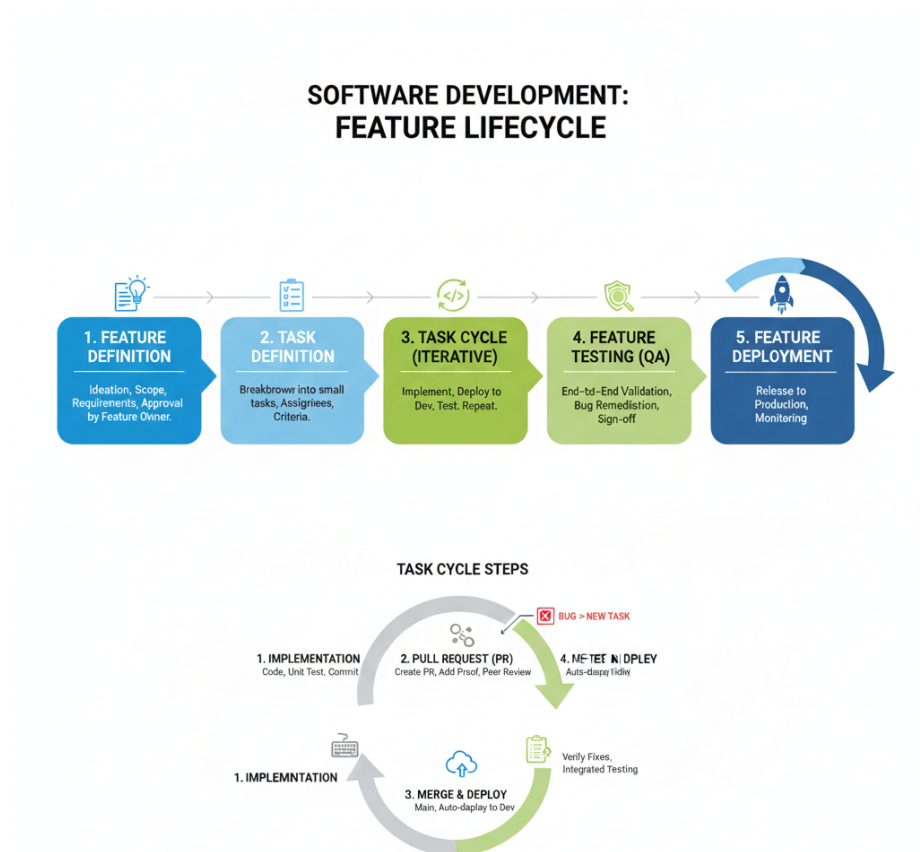


Figure 1: Feature Lifecycle Diagram

Task Lifecycle

The Task Cycle is the iterative loop that developers follow to build, review, and integrate specific pieces of the feature.

Stage	Description
Task Definition	Detailed plan for a single developer's work (e.g., build API endpoint).
Task Implementation	Coding, local testing, code review, and merging into the main branch.
Task Deployment	Automatic deployment to the Dev/Integration environment upon merge.
Task Testing	Re-testing the deployed code in QA (as part of a pre-release).

Feature Definition

The Feature Definition stage establishes the ‘Why’ and ‘What’ of a new capability, ensuring clear goals and requirements before any development begins.

Key Stakeholders Involved

The following roles are essential for defining and approving a feature:

- **Architect:** Ensures the feature aligns with the overall system architecture and technical vision.
- **UI/UX Architect:** Defines the user experience, wireframes, and interface design.
- **Feature Owner (Product Owner):** Responsible for the feature's success, prioritizing the work, and defining the acceptance criteria.

Core Questions and Requirements

Question	Definition/Purpose	Example
Who is the feature for?	Defines the target user segment (e.g., administrators, end-users, mobile users).	“For logged-in users who have purchased a premium subscription.”
What is the feature trying to accomplish?	Defines the business value and objective.	“Allow users to securely update their payment information to reduce payment failure rates.”
Mock-ups for User Interface(s)	Visual representations of the new screens or elements.	Link to Figma or Sketch designs for the “Payment Settings” page.

Question	Definition/Purpose	Example
What data is required?	Outlines all necessary data fields and flows.	Input/User Facing: Credit Card Number (tokenized), Expiration Date, Name on Card. Processing: Tokenization service call, validation rules, API payload format.
What is the Acceptance Criteria?	A list of verifiable conditions that must be met for the feature to be considered complete and successful from a user/business perspective.	“1. A user can successfully save a new credit card.” “2. An error message appears if the card is expired.”
What should be saved for future features?	Identifies non-immediate data needs (e.g., audit logs, historical usage data).	“Save a record of the old and new card token for audit history, even if not immediately displayed.”

Approval

A feature is considered officially defined and ready to proceed when the **Feature Owner** provides final sign-off, confirming all requirements and criteria are documented.

Feature Testing & Deployment

This is the final verification stage where the full feature is validated, approved, and released to the end-users.

Test Deployment (QA Release)

Before final deployment, the completed feature set is formally released to QA for end-to-end validation.

- **Completion Criteria:** This stage begins when **all tasks for a feature are completed and merged**.
- **Definition of Done:** What defines “Done” needs to be explicitly defined by the organization. This must include consideration for any “**known Defects**”—bugs that the business accepts as non-critical for the initial release.

- **Release Artifact:** A new feature release is created and set as a **pre-release** in GitHub (or similar VCS) that includes all the code to be released.
- **Deployment:** The code tied to that pre-release tag is then deployed to the **QA Environment**.

Feature Testing

The objective is to validate the entire user journey and feature functionality in an environment that mimics production.

- **Action:** Once deployed to QA, the whole application is re-tested (often via regression or smoke tests).
- **Targeted Testing:** Depending on the scope, resources, and risk of the feature, there may be targeted testing of the most critical parts or dependent features.
- **Final Definition of Done:** Re-evaluation of the “Done” criteria. All Acceptance Criteria from the Feature Definition must be satisfied.

Defect Remediation and Approval

Bugs found during Feature Testing must be addressed quickly before production deployment.

- **New Bug Task:** Any defect found should result in a new **Bug task** being created and assigned to a developer.
- **Implementation:** The fix is implemented in a new branch, following the full **Task Implementation** steps (PR, review, merge).
- **Recycle:** Once merged, the new fix is included in the current QA release/pre-release, and **Feature Testing is repeated** to verify the fix and ensure no regressions.

Deployment Approval

This is the final gate before shipping the code to production.

- **Sign-Off:** Once all defects are either **fixed** or formally accepted (put into the backlog as “known defects” with Feature Owner sign-off), final approval is required.
- **Scheduling:** Deployment to production is then scheduled according to organizational policy (e.g., maintenance window, continuous delivery pipeline).

Feature Deployment (Production)

The final step is releasing the approved, tested code to the live environment.

- **Action:** The approved release tag is deployed to the **Production Environment**.
- **Validation:** Post-deployment smoke testing is performed to ensure the application is functioning correctly in a live setting.
- **Monitoring:** Continuous monitoring is enabled to catch any unexpected behavior or performance issues.

Task Integration

This phase ensures that the newly merged code is available for integration testing and is verified against the original task requirements.

Task Deployment to Dev/Integration

The goal is to provide a rapidly updated environment for developers to test their changes together.

- **Trigger:** Once the code is merged to the **main** or **master** branch.
- **Action:** The tip of the **main/master** branch is automatically deployed to the **Dev/Integration** environment.
- **Purpose:** Allows developers to immediately see their code running in a shared environment and test integration with other newly merged tasks.

Task Testing in QA Environment

Formal testing of the task happens in the QA environment as part of a grouped release process.

- **Pre-Release Trigger:** When a group of completed tasks (or the entire feature) is ready for QA, a new **pre-release tag** is created in the repository (e.g., GitHub).
- **Deployment:** The code that is part of that pre-release is then automatically deployed to the **QA Environment**.
- **Action:** The original developer (or a dedicated Tester) re-tests the task in the stable QA environment using the steps and criteria defined in the Task Definition.

Defect Handling

Any defects (bugs) found during Task Testing must be handled systematically:

1. **Triage:** Determine if the defect is a new, unrelated bug, or a direct failure of the current task implementation.
2. **Scheduling:** Defects determined to be low priority or outside the current scope should be scheduled as a **new task in a different development cycle**.

3. **Immediate Fix:** If the defect must be fixed immediately, the developer uses the original branch (if feasible) or a new fix branch to implement the correction.
4. **New PR:** A new Pull Request should be created, reviewed, merged, and re-tested following the **Task Implementation** steps.