DAY 12: RTM Jira + Zephyr

Presentation.pptx

Learning Objectives

By the end of this guide, you will be able to:

- Understand the structure and workflow of using Zephyr with Jira for Requirements Traceability Management (RTM).
- Create test cases, link them to requirements, and track execution results.
- Use Zephyr’s reporting features to maintain traceability and ensure test coverage.
- Integrate Zephyr workflows into Agile and Scrum methodologies.

Introduction to Zephyr for Jira

Zephyr is an add-on that enhances Jira by providing dedicated test management capabilities, making it easy for teams to manage and execute tests directly within Jira. Zephyr integrates seamlessly with Jira, enabling testers to manage the testing lifecycle efficiently.

Key features of Zephyr for Jira include:

- Test Cases: Detailed descriptions of tests, including the testing steps, preconditions, and expected outcomes. These represent individual testing scenarios.
- Test Cycles: Groups of test cases that are planned to be executed together in a specific testing phase (e.g., Sprint Testing, Regression Testing).
- Test Executions: Tracking the actual performance of a test case, showing its status (e.g., Pass, Fail, Work in Progress) after execution.
- Metrics and Reporting: Provides detailed insights into the testing process, such as progress, test coverage, and defect status, helping the team evaluate the application's quality.

Setting Up a Manual Testing Process in Jira with Zephyr

Here’s a step-by-step process for setting up manual testing in Jira, especially with Zephyr integration:

Step 1: Configuring Jira

1. Create a Project: Start by setting up a project dedicated to testing or integrating testing into an existing software development project. Projects in Jira help organize issues (like test cases, bugs, or tasks).
2. Define Issue Types: Jira supports various issue types (e.g., Bug, Task, Story). When using Zephyr, you'll have access to "Test Case" as a predefined issue type. If Zephyr is not installed, you can create a custom issue type for test cases.

Step 2: Creating Test Cases

1. Add Test Cases: Using the "Test Case" issue type in Zephyr, you can create detailed test cases with fields for test steps, expected results, and any preconditions.
2. Organize in Test Cycles: Group related test cases into test cycles to plan which tests to run in different phases (e.g., during development sprints or in a dedicated regression testing cycle).

Step 3: Test Execution

1. Run Test Cases: Execute your test cases by changing their status (e.g., Pass, Fail, WIP). Zephyr allows you to track the test execution status within Jira.
2. Defect Reporting: If any issues arise during test execution, they can be reported as bugs within Jira. These defects can be linked directly to the test cases that caused the failure, allowing traceability between test cases and issues.

Step 4: Monitoring and Reporting

1. Use Dashboards: Create Jira dashboards to monitor the status of test cycles, track progress, and review any defects found.
2. Generate Reports: Zephyr's built-in reporting tools allow teams to assess the overall quality of the software, test coverage, and testing progress, providing real-time insights into testing activities.

Step 2: Integrating Zephyr with Jira

1. Install Zephyr: Zephyr can be installed from the Atlassian Marketplace. Once installed, Zephyr test management features will become accessible within Jira, enhancing its capabilities for manual testing.
2. Configure Zephyr: Set up key configurations such as:
    - Test statuses (e.g., Pass, Fail, Blocked).
    - Test step fields for defining the details of test cases.
    - Tailor Zephyr settings based on your project's needs, such as adding custom fields or statuses relevant to your testing process.

Step 3: Creating Test Cases

1. Create Test Cases in Zephyr: With Zephyr installed, create new test cases by:
   - Defining test steps, expected results, and any necessary preconditions.
   - Attaching relevant documentation, screenshots, or additional reference materials to provide context for the test cases.
2. Organize Test Cases: Use folders, components, or labels to categorize test cases for better management. This helps testers quickly find and execute relevant test cases based on different testing phases or modules.

Step 4: Planning Test Cycles

1. Create Test Cycles: A test cycle defines a specific testing phase (e.g., Sprint Testing, Regression Testing):
   - Assign test cases to the cycle based on what needs to be tested in that phase.
   - Set timelines for executing these test cases and designate testers responsible for each case.
2. Execute Tests:
   - Testers manually execute test cases.
   - Record the test results in Zephyr (e.g., Pass, Fail).
   - Log any defects directly from Zephyr and link them to the relevant test steps for traceability.

Step 5: Tracking Progress and Reporting

1. Monitor Test Execution:
   - Utilize Zephyr's dashboards and built-in reports to get real-time insights into the status of test executions.
   - View overall progress, track defect counts, and identify test cases that are incomplete or failed.
2. Generate Reports:
   - Generate detailed reports on test coverage, defect density, and progress.
   - Share these reports with stakeholders to provide an overview of the testing phase and guide decision-making.

Best Practices for Manual Testing with Jira and Zephyr

- Regularly Review and Update Test Cases: Ensure that test cases are always up to date to reflect changes in the application, preventing test cases from becoming outdated or irrelevant.
- Use Labels and Components Wisely: Properly organize test cases using labels or components, making it easier to filter and manage large numbers of cases.

Leverage Integration Features: Link defects directly to test cases, making it easier to trace bugs back to their origin and to determine their impact.

RTM :

RTM stands for Requirements Traceability Matrix, a document used in software development to ensure that all requirements are covered throughout the project's lifecycle, from design and development to testing and delivery. It’s a tool to trace and manage project requirements, helping to verify that all requirements have been met and making it easier to identify gaps or changes.

Types of RTM

1. Forward Traceability Matrix
   Purpose: Tracks requirements from their origin to the final deliverables (test cases). This ensures that each requirement is verified by a corresponding test case and that development activities align with the project requirements.
   Example:
   A requirement to create a login feature is documented in the matrix and traced forward to see if there is a corresponding test case that validates it and if that feature was implemented in the final product.
2. Backward Traceability Matrix
   Purpose: Tracks each deliverable or test case back to its original requirement. This ensures that only necessary functionality, as defined by requirements, has been implemented, preventing scope creep.
   Example:
   If there is a new test case or feature, it is traced back to verify if there is a requirement for it. For instance, a "Remember Me" feature in the login module is checked to ensure it was a specified requirement.
3. Bidirectional Traceability Matrix
   Purpose: Combines forward and backward traceability, providing a full traceability view of the project. It allows tracking both forward (from requirements to test cases) and backward (from test cases back to requirements), ensuring complete coverage and validating that all work aligns with the requirements.
   Example:
   A requirement for a search functionality is traced forward to see if it was developed and tested. Additionally, every test case for search functionality is checked backward to ensure it is linked to an initial requirement.

Requirement ID	**Requirement**	**Design Specification**	**Code Module**	**Test Ca**	**Test Description**	**Status**	**Comments**

	Description			se ID			
REQ-001	User Registration	DS-001	Register.java	TC-001	Verify user can register with valid details	Passed	Successful
REQ-002	User Login	DS-002	Login.java	TC-002	Confirm user can log in with valid credentials	Passed	Successful
REQ-003	Product Search	DS-003	Search.java	TC-003	Check search functionality for accurate results	Passed	Needs search optimization
REQ-004	Add to Cart	DS-004	Cart.java	TC-004	Verify product adds to cart successfully	Passed	N/A

REQ-005	Update Cart Quantity	DS-005	CartUpdate.java	TC-005	Ensure users can update item quantity in cart	Passed	Successful
REQ-006	Remove from Cart	DS-006	CartRemove.java	TC-006	Confirm item removes from cart correctly	Passed	N/A
REQ-007	Checkout Process	DS-007	Checkout.java	TC-007	Validate checkout with item and address details	Passed	N/A
REQ-008	Payment Gateway Integration	DS-008	Payment.java	TC-008	Confirm payment integration with multiple options	Failed	Issues with payment gateway

REQ-009	Order Confirmation	DS-009	OrderConfirm.java	TC-009	Check order confirmation is sent after payment	Passed	Email confirmation working
REQ-010	View Order History	DS-010	OrderHistory.java	TC-010	Verify user can view past orders	Passed	N/A
REQ-011	Add Product Reviews and Ratings	DS-011	Review.java	TC-011	Ensure users can submit reviews	Passed	N/A
REQ-012	User Profile Update	DS-012	ProfileUpdate.java	TC-012	Confirm users can update profile details	Passed	N/A
REQ-013	Wishlist Management	DS-013	Wishlist.java	TC-013	Verify users can add/remove	Passed	N/A

					items from wishlist		
REQ-014	Push Notifications for Order Updates	DS-014	Notification.java	TC-014	Check users receive push notifications	Pending	Pending notification settings
REQ-015	Address Management	DS-015	Address.java	TC-015	Confirm user can add/edit/delete address details	Passed	N/A

Conclusion

Integrating Jira with Zephyr simplifies and enhances the manual testing process by offering tools for better planning, execution, and reporting. This integration strengthens collaboration between QA teams and developers, improves traceability of defects, and ultimately leads to higher-quality software. By following these steps and best practices, teams can make their manual testing efforts more efficient and transparent.