Managing Test Libraries

A test library (also known as a test repository) is where test cases are created and stored. You can share and reuse test cases across multiple Jira development projects (libraries). Within a test library, you can organize test cases using folders, labels, and custom fields.

Benefits of creating and managing test libraries include the ability to:

- Use case A: Design and use test cases in the same development project
- Use case B: Reuse test cases across different development projects
- Use case C: Set up a Jira project to act as a library for test cases

Managing Folders

As you increase the number of test cases in your test library, you will eventually need to organize them more efficiently. Test cases can be organized and maintained through a hierarchical folder structure in a single library for quick access.
As a best practice for building your test library, consider having each folder cover a specific functional area, screen, component, requirement, or test type, such as UAT tests, integration tests, or system-level tests.

Reusing Test Cases

To avoid losing speed or sacrificing quality, software testing teams must continually be on the lookout for smarter ways to plan, design, and execute test cases. Adopting modularisation, parameterisation, and data-driven testing techniques to test-case design can help you establish large-scale, reusable test libraries. We recommend starting with small, manageable chunks and aiming to build a modular, flexible test library that ensures reusability over time. Adopting a modular test-case design with building blocks can boost productivity and ensure adequate coverage of an application's functionality.

There are many benefits to this approach, including:

- Reusable large-scale test libraries
- Reduced redundancy
- Maximized test coverage
- Reduced maintenance efforts
- Increased efficiency and productivity

Modularisation

One effective way to boost reusability for large-scale test-cases libraries is to adopt a modular test-case design approach. Instead of duplicating or copying and pasting test cases or steps, this approach emphasizes breaking them down into small, reusable pieces and then recombining the tests to achieve larger end-to-end testing scenarios.

The most valuable test cases are those which are broadly applicable and reusable across different cycles and releases. Test engineers should devote serious thought into how to design test cases to be as specific as they need to be but also generic enough to be reused in different situations with different data inputs.

Testers can break down test cases into logical, manageable functions or modules. These modules are isolated to create independent tests that can then be recomposed or reused to make larger master test cases to achieve complex end-to-end testing scenarios. Test Management for Jira gives you the ability to nest child modules under master test cases, even with multiple levels. Each set of steps will be unfolded during the test execution (with the parameter replaced).

To begin a modular strategy, follow the steps below:

1. Understand the workflow of the application before you start to identify which modules need to be created.
2. Map out steps that are repeated on a page or across multiple pages.
3. Break down the repeated steps into bite-sized reusable test cases.
4. Use a skeleton master test case to call out the test cases.

Reusing Modular Test Cases

1. From the Test Script tab of any test case, click Call to Test to reuse existing modular test cases.
2. Browse and select the desired test cases, then click Add.

3. Click Show Steps to view the steps.
The reusable steps appear in the expanded step view.

In the Test Player screen, you'll notice all steps are unfolded in a flat step list.
Parameterisation

Test parameterisation allows you to call out test cases passing a set of different values. In essence, parameters are used to unlock more flexibility when calling or invoking reusable test cases. And if you are reusing a modular test case, the master test case can invoke a reusable modular test case by passing a set of parameters, which can affect how input or output data are displayed or processed during test execution. Eventually, parameters can contain default values, and the tester can change each of the default parameters according to the context of the test at hand.

Working with Parameterisation

1. Create a test case and enable the parameter functionality on the Test Script tab.
2. Add parameters to the test case. You can also define default values. In the steps, you can reference parameters by typing a { brace (curly bracket), which triggers a parameters drop-down list.

3. In the master test case (the test case calling or invoking reusable modules), click Call to Test to reuse existing modular test cases.
4. Select the desired test cases, then click Add.

The modular test case displays in a special step type called Call to Test.

5. Fill in the desired values for the parameters.
6. Click Show steps to view the steps.
The reusable steps appear in the expanded step view.

The variable placeholders point to the parameters. During the test execution, these placeholders will be replaced by the values in the parameters field.
Data-Driven Testing

Data-driven testing (DDT) is an approach to test design where the test data (input and output values) is separated from the actual test case. The test data can be stored in one or more central data sources and shared across different test cases. By storing your test data in a central repository (local storage, excel spreadsheet, XML file, or SQL database), you can run the same test with a new set of data each time, avoiding redundant design and execution of repetitive tests cases. Toward the end of the test cycle, you can store test data to provide a clear audit trail of what was and was not covered by a test.

Test cases are commonly designed using variables in test steps. Variable placeholders receive values from a data source instead of using hard-coded or fixed values. Each time you execute the test case, the variable is replaced with the test data in the corresponding data source, reducing the time and effort necessary to create new test cases. With this approach, a new test case simply requires a new row of test data in the data source. Additionally, this approach provides a clear separation of testing logic from the actual test data, which reduces maintenance efforts (i.e., any changes to test steps do not affect the test data and vice-versa).

In a nutshell, DDT can be visualised as follows:
When designing test cases using the DDT approach, the focus should be on mapping the test data and the different combinations and variations (negative and positive) to properly meet the business requirements and ensure adequate test coverage. By doing so, when executing test cases using Test Management for Jira, the variable placeholders in test steps are replaced by the values available in the test-data table, ensuring complete reusability of the test steps.

DDT can be used with manual test scenarios, enabling manual test scripts to run together alongside their respective data sets. In DDT, the following operations are performed in a loop for each row of test data in the data source:

- Test data is retrieved
- Test data is input into the system under test (SUT) and simulates other actions
- Expected values are verified
- Test execution continues with the next set of test data

**Data-driven testing in action**

Depending on the business rules you have to validate during the test case design, you may need to repeat the test case several times with different test data to ensure that the application works as expected for various input values. For example, if you have to design test cases to test the search and book a flight functionality of an airline company, you’ll need to check different combinations of input values for the search screen (i.e., from, to, depart, return, class, etc). To prevent duplication, along with the test case you can provide a data source with test data to map each combination you need to address to leverage the coverage, as seen in the image below:
When designing test cases using the DDT approach, the focus should be in mapping the test data and its different combinations and variations (negative and positive) to meet the requirements and business rules and ensure adequate test coverage. By doing so, during the test execution, the variable placeholders in the test step level are replaced by the values available in the test data table, ensuring complete reusability of the steps instead of using hard-coded or fixed values as displayed in the image below:
When designing test cases using the DDT approach, the focus should be on mapping the test data and the different combinations and variations (negative and positive) to properly meet the business requirements and ensure adequate test coverage. By doing so, when executing test cases using Test Management for Jira, the variable placeholders in test steps are replaced by the values available in the test-data table, ensuring complete reusability of the test steps.

DDT can be used with manual test scenarios, enabling manual test scripts to run together alongside their respective data sets. In DDT, the following operations are performed in a loop for each row of test data in the data source:

- Test data is retrieved
- Test data is input into the system under test (SUT) and simulates other actions
- Expected values are verified
- Test execution continues with the next set of test data

**Benefits of data-driven testing**

Adopting DDT is a way of ensuring repeatability and reusability of existing test cases. DDT leverages test coverage by feeding different types of test data – either valid or invalid (positive and negative test cases). As a result, you can benefit from a dramatic reduction of redundant test cases and bugs in production. Other benefits of adopting DDT include:

- Increased test coverage
- Reduction in time spent designing and maintaining test cases
- Repeatability and reusability of existing test cases
- Reduction in the number of test cases

**Working with Data-Driven-Testing (DDT)**

1. Create a test case and enable the test data functionality in the Test Script tab.
2. Add columns (variables) to the test-data table.
   In the steps, you can reference a column (variable) by typing a { brace (curly bracket), which triggers a drop-down list with column options.

   When the test case is ready to be executed in the Test Player, the test steps are repeated to ensure complete coverage.

   ![Test Player screen with test data and steps]

   In the Test Player screen, you'll notice all steps are unfolded in a flat step list. Also, each step will be unfolded during the test execution with the parameter replaced with the values passed by the master test case. Bear in mind, each row in the test data table will create a new group of steps during the unfolding procedure as shown in the image below.
Using datasets

Datasets allow you to define standardized test data parameters that can be re-used in test cases across the Project on the test scripts. Datasets ensure consistency across test cases and prevent data duplication.

1. Create datasets with the desired set of reusable data.

2. When creating data columns, select and add the desired dataset to ensure consistency in the data input.
3. When populating the test data table, select the desired values from the datasets to prevent data duplication and ensure consistency.
Next Steps

- Start your free 30-day trial now.

Have a question for us?

- Please submit a support request. We are happy to help!