Designing Test Cases

The following steps illustrate the basics of test-case design.

1. Open a project and click the Tests view.

2. Click New.

The Create Test Case screen appears.

3. On the Details tab, fill in the standard and custom fields.
4. Click the Test Script tab, and enter the steps.

5. Use the remaining tabs to complete the described tasks:
   - The Execution tab allows you to track previous test execution results.
   - The Traceability tab allows you to link Issues, Confluence pages, and Web links to the test case.
The Attachments tab, you can attach files to the test case.
The Comments tab allows you to fill in comments for the test case.
The History tab allows you to track changes made to the test case.

6. Click Save.
The test case is saved.

Advanced Techniques

The concepts and procedures for test-case design outlined in this section are more complex but, when used appropriately, can lead to even greater testing efficiencies and results.

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 that you break them down into small, reusable pieces and then recombine the tests to achieve larger end-to-end testing scenarios. Each step reused will be unfolded during the test execution.

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 recombined 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 which steps 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.

Add Existing Test Cases

The modular test cases appear in a special step type called Call to Test.

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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Input the username</th>
<th>TEST DATA</th>
<th>Click to type the test data</th>
<th>Expected Result</th>
<th>Click to type the expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Click on the login button</td>
<td>TEST DATA</td>
<td>Click to type the test data</td>
<td>Expected Result</td>
<td>Logged in successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each step will be unfolded during the test execution (with the parameter replaced with the values passed by the master test case).

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.
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.

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 test cases. Toward the end of the test cycle, you can store test data to provide a clear audit trail of what was and wasn’t 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

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.
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.

Versioning

Version control is enabled by default and allows teams to collaborate and manage multiple versions of test cases simultaneously while maintaining data integrity and providing a history of changes. Versions can be viewed together and compared for audit purposes.
Changes to testing details only affect test executions using the current version. Changes to test steps do not affect test executions automatically to preserve the results history. However, the following notification appears on the Test Player screen: This test script is outdated because the test case has been changed. If you choose to update the test script, the steps are synced and updated.

You cannot create a new version directly from an older version, but you can make changes to older versions. Also, by default, all lists, grids, searches, queries, etc. display the latest version of test cases.

**Working with Versioning**

1. Open a project and click the Tests view.
2. Click New. The *Create Test Case* screen appears.
3. Add a name, objective and precondition.
4. Click Save. A new version of the test case is created (1.0).
5. Click New Version.

A prompt appears.

6. Click New Version on the prompt.

A new version of the test case (2.0) is created. You can click the dropdown with the version number to navigate to different versions.
7. Click the History tab, where versions can be viewed and compared for auditing purposes.

8. Select the versions you want to compare, then click Compare Versions.

The Compare Versions screen appears.
Gherkin Behaviour-Driven Development

Writing Behaviour-Driven Development (BDD) test cases is easy. Test Management for Jira allows you to create a BDD test case in Jira and export it to a feature file. The feature file can be used as a reference for writing unit tests that are necessary for it to run.

BDD test cases are written in Gherkin syntax. Here is a list of supported and not supported keywords:

<table>
<thead>
<tr>
<th>SUPPORTED</th>
<th>NOT SUPPORTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given</td>
<td>Feature</td>
</tr>
<tr>
<td>When</td>
<td>Scenario</td>
</tr>
<tr>
<td>Then</td>
<td>Background</td>
</tr>
<tr>
<td>And</td>
<td>Scenario Outline</td>
</tr>
<tr>
<td>But</td>
<td></td>
</tr>
</tbody>
</table>
Supported keywords such as *Given*, *When*, and *Then* are highlighted as you type your test case.

**Working with BDD Gherkin Scripts**

1. Open a test case, and click the Test Script tab.
2. Select BDD - Gherkin from the Type dropdown.
   The Gherkin editor appears.
3. Write your test case. Remember not to include Feature or Scenario.
4. Click Save when you are finished, and close the test case.
5. Select Export Feature Files (BDD - Gherkin) from the More to download the feature file for your new test case (or the selected test cases).
The zip file downloads to your computer. Developers using your company's supported BDD tool can reference this information when writing automated tests related to the test case. For more information about test automation, learn more here: API and Test Automation

Next Steps

- Start your free 30-day trial now.

Have a question for us?

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