

Planning tests from a "methodology" perspective

- [Test planning approaches considering the software development methodology](#)
 - [Agile](#)
 - [Waterfall](#)
 - [Iterative Waterfall](#)
- [Examples using Dynamic Test Plans](#)
 - [Tests that cover requirements for a specific sprint](#)
 - [Tests that cover requirements assigned to the current open/active sprints](#)
 - [Tests that cover requirements for the current version](#)
 - [Tests that cover requirements from previous versions](#)
- [Learn more](#)

Test planning approaches considering the software development methodology

Agile

Scrum

If you're following Agile, particularly Scrum, you'll have sprints of one or more weeks. Each sprint represents one iteration until, at the end, you have a shippable product. Testing can occur during the sprint or more closely to the end. In any case, you can have a Test Plan per sprint to track the results of the Tests you want to execute, including the ones that validate the features implemented in that sprint. Besides this, you may also want to have a Test Plan for regression testing to ensure you're staying within that sprint.



Please note

A sprint is a more manageable way of dividing the complexity of an entire release. A sprint has a short, well-defined period.

Kanban

In Kanban, you're limiting the amount of WIP (work-in-progress) issues in a given state, namely, the ones in testing. Because you're not dividing your release into several periods, as you do with Scrum, you can manage it as if the period would embrace the time frame of the release.

Thus, you can have a Test Plan to track the Tests related to the features being addressed on the Kanban board. You may complement it with additional Test Plans for tracking the regression testing. It may be an excellent approach to have the Test Plan with regression tests, or smoke tests, as an independent Test Plan, so it is more apparent if regression is occurring.

Waterfall

In this scenario, testing is done as a separate phase at the end of development. Bugs may be found during testing, which will be directed back to requirements analysis/development. You wait for changes/fixes and then test it again.

Having a Test Plan assigned to the version currently being developed to group the results from multiple testing cycles may be enough.

Iterative Waterfall

In this approach, a release is split into several internal mini-releases, with subsets of the features implemented. It is assumed that some of these features may not be complete/stable/finished.

Each intermediate release may have one specific Test Plan to track the Tests and their results. The Test Executions made in the context of each of these Test Plans can also be associated with a broader Test Plan, so their results would also get reflected in a broader sense.

In sum, you can create one Test Plan for tracking all the Tests you want to validate in that version, along with additional Test Plans, one per iteration /intermediate release. Then, you must ensure that all Test Executions created in the context of your intermediate release's Test Plan are also reflected in the "master" Test Plan.

Examples using Dynamic Test Plans



Dynamic Test Plan is a Xray Enterprise Feature



Dynamic Test Plans is a feature of [Xray Enterprise](#). If you do not have Xray Enterprise installed, the **Dynamic Test Plans are not available** in the Test Plan issue, and it is impossible to configure dynamic test lists. When installing Xray Enterprise for the first time, please re-enable Xray to load all properties from Xray Enterprise.

Depending on the "methodology" used, your team may have different needs or approaches regarding Tests.

Tests that cover requirements for a specific sprint

We will share an example that can be extended for other methodologies using different JQL Queries. Still, for this one, we are considering the Agile Scrum methodology.

Context

- A tester wants to have a Dynamic Test Plan with all the Tests that matter for the sprint.



JQL features in Jira Cloud

JQL in Jira Cloud has some limitations compared with Jira Data Center; Atlassian has been improving JQL support in Cloud but there are still some gaps; some of these can be overcome with other Jira apps as seen ahead. To achieve better results, we advise you use [ScriptRunner app](#) (you may also be able to implement this using another app with similar capabilities) to create enhanced filters.

How to

To be able to implement this use case, we'll use ScriptRunner Enhanced Search feature of the [ScriptRunner app](#) (you may also be able to implement this using another app with similar capabilities).

In this example, we have an active Sprint named "CALC Sprint 1" with 2 user stories, each one covered by some Test(s).

The screenshot displays the Jira Software interface for a project named 'Calculator'. The main view is a Kanban board for 'CALC Sprint 1'. The board is divided into three columns: 'TO DO 2', 'IN PROGRESS 1', and 'DONE'. In the 'TO DO' column, there are two items: 'dummy story' (CALC-15) and 'dummy tp' (CALC-17). In the 'IN PROGRESS' column, there is one item: 'As a user, I can login the application' (CALC-152). The right sidebar shows the details for the selected issue, CALC-152, including its description, a 'Description' field, and a 'Linked issues' section showing a linked issue 'CALC-153 usability test'.

1. In **Apps > ScriptRunner Enhanced Search** create a JQL Filter named "tests_for_calc_sprint1" that lists all Tests covering user stories in the sprint named "CALC Sprint 1".

- **Sample JQL**

```
issueFunction in linkedIssuesOf("project = CALC and issuetype = \"Story\" and sprint in openSprints()
\", \"is tested by\")
```

The screenshot shows the Jira ScriptRunner Enhanced Search interface. On the left, there's a sidebar with 'SAVED FILTERS' including 'tests_for_calc_sprint1'. The main area shows the filter details for 'tests_for_calc_sprint1' with the JQL query: `issueFunction in linkedIssuesOf("project = CALC and issuetype = \"Story\" and sprint in openSprints()\", \"is tested by\")`. Below the query, there's a table of results with columns: Key, Assignee, Created, Issue Type, Priority, Reporter, Status, Summary, and Updated. Two results are shown: CALC-153 (usability test) and CALC-16 (simple integer addition).

2. We then create a Dynamic Test Plan based on that query. To do so, we create a Test Plan as usual and then configure it use a saved filter.

The screenshot shows the 'dynamic test plan' configuration page for project CALC-323. It includes tabs for 'Attach', 'Create subtask', 'Link issue', 'Tests', 'Scripted Fields', and 'Risk assessment'. The 'Tests' tab is active, showing 'Create Test Execution' and 'Trigger Build' buttons. A 'FILTER QUERY' section is visible with a 'View on board' link and a 'Configure' button highlighted with a red box.

The screenshot shows the 'Configure Test Plan CALC-323' dialog box. It explains that the option determines which issues will appear on the Test Plan. There are two radio buttons: 'Static' and 'Saved Filter'. The 'Saved Filter' option is selected. Below it, a dropdown menu shows 'tests_for_calc_sprint1' with a 'New Filter' link. A 'View Filter Query' link is also present. The 'Filter Query' section shows the query: `issue in (10959,11396)`. At the bottom, there are 'Save' and 'Cancel' buttons.

As you can see, the Tests are automatically added to the Test Plan; if they are implicitly removed from the filter they will be automatically removed from the Dynamic Test Plan.

We added the column "Linked Issues" on the Tests panel to show all linked issues, which include, among others, the user stories that each Test cover.

dynamic test plan

[Attach](#)
[Create subtask](#)
[Link issue](#)
[Tests](#)
[Scripted Fields](#)
[Risk assessment](#)

Description

Example of a dynamic test plan

Tests

[Tests](#)
[Test Executions](#)

Create Test Execution

Trigger Build

FILTER QUERY

View on board

Overall Execution Status

All Environments, final status

2 TO DO

TOTAL TESTS: 2

Only My Issues

Filters

10

Columns

Key	Summary	Assignee	#Test Executions	Dataset	Priority	Linked Issues	Latest Status	Actions
<input type="checkbox"/> CALC-153	usability test		0		=	CALC-158, CALC-152	TO DO	
<input type="checkbox"/> CALC-16	simple integer addition		0		=	CALC-15	TO DO	
Prev 1 Next								Total 2 issues

Tests that cover requirements assigned to the current open/active sprints

Context

- A tester wants to create a Test Plan with all the tests that cover requirements in the current open/active sprint(s).

How to

To be able to implement this use case, we'll use ScriptRunner Enhanced Search feature of the [ScriptRunner app](#) (you may also be able to implement this using another app with similar capabilities).

- In **Apps > ScriptRunner Enhanced Search** create a filter (e.g., "tests_for_current_sprints") for the following JQL query.

a. Sample JQL

```
issueFunction in linkedIssuesOf("project = CALC and issuetype = \"Story\" and sprint in openSprints() ", "is tested by")
```

ScriptRunner Enhanced Search

JQL Sync Status: FULLY SYNCED

SAVED FILTERS

Sort by: Alphabetical

risk_medium_stories

tests_for_current_sprints

tests_for_current_sprints Sync Filter

IssueFunction in linkedIssuesOf("project = CALC and issuetype = \"Story\" and sprint in openSprints() ", "is tested by")

Type your JQL query or use "+" to insert helpers and functions

Some keywords in Enhanced Search will be removed soon. Learn more and see which ones in our documentation.

1-2 of 2

Results

Key	Assignee	Created	Issue Type	Priority	Reporter	Status	Summary	Updated
CALC-153		02/Sep/22 2:26 pm	Story	=	Sérgio Freire	TO DO	usability test	16/Aug/23 10:41 am
CALC-16		21/Mar/22 11:12 am	Story	=	Sérgio Freire	TO DO	simple integer addition	21/Mar/22 11:12 am

- Use the previous saved filter on the Test Plan configuration, as a regular Jira filter.

Tests that cover requirements for the current version

Context

- A tester wants to create a Test Plan with all the tests that cover requirements assigned to the current version.

How to

To be able to implement this use case, we'll use ScriptRunner Enhanced Search feature of the [ScriptRunner app](#) (you may also be able to implement this using another app with similar capabilities).

1. In **Apps > ScriptRunner Enhanced Search** create a filter (e.g., "tests_for_calc_v2") for the following JQL query.

a. **Sample JQL**

```
issueFunction in linkedIssuesOf('project = CALC and issuetype in ("Story", "Epic")) and fixVersion = "v2.0"', "is tested by")
```

b. ... or ...

c. **Sample JQL**

```
issueFunction in linkedIssuesOf('project = CALC and issuetype in ("Story", "Epic") and fixVersion = earliestUnreleasedVersion(CALC) ', "is tested by")
```

d.

The screenshot shows the Jira ScriptRunner Enhanced Search interface. On the left, there's a sidebar with 'SAVED FILTERS' including 'risk_medium_stories' and 'tests_for_calc_v2'. The main area shows the filter 'tests_for_calc_v2' with its JQL query: 'issueFunction in linkedIssuesOf('project = CALC and issuetype in ("Story", "Epic") and fixVersion = "v2.0"', "is tested by")'. Below the query, there's a table of results with columns: Key, Assignee, Created, Issue Type, Priority, Reporter, Status, Summary, and Updated. The first result is 'CALC-153' created on '02/Sep/22 2:26 pm' by 'Sergio Freire' with status 'TO DO' and summary 'usability test'.

2. Use the previous saved filter on the Test Plan configuration, as a regular Jira filter.

Tests that cover requirements from previous versions

Context

- A tester wants to create a Test Plan with all the tests that cover requirements from previous versions.

How to

To be able to implement this use case, we'll use ScriptRunner Enhanced Search feature of the [ScriptRunner app](#) (you may also be able to implement this using another app with similar capabilities).

This use case is similar to the previous example, we just need to slightly adjust the query.

1. In **Apps > ScriptRunner Enhanced Search** create a filter (e.g., "tests_for_previous_releases") for the following JQL query.

a. **Sample JQL**

```
issueFunction in linkedIssuesOf('project = CALC and issuetype in ("Story", "Epic")) and fixVersion in ("1.0", "1.1", "1.2")', "is tested by")
```

b. ... or ...

c. Sample JQL

```
issueFunction in linkedIssuesOf('project = CALC and issuetype in ("Story", "Epic")) and fixVersion in releasedVersions()', "is tested by")
```

The screenshot shows the 'ScriptRunner Enhanced Search' interface. On the left, under 'SAVED FILTERS', there are two filters: 'tests_for_calc_previous...' and 'tests_for_calc_sprint1'. The main area shows the filter 'tests_for_calc_previous_releases' with a JQL query: `issueFunction in linkedIssuesOf('project = CALC and issuetype in ("Story", "Epic")) and fixVersion = earliest(UnreleasedVersion(CALC), "is tested by")`. Below the query, it says 'Type your JQL query or use "+" to insert helpers and functions'. A message states: 'Some keywords in Enhanced Search will be removed soon. Learn more and see which ones in our documentation.' Below that, it shows '1-1 of 1 12' and a 'Choose Columns' dropdown. The 'Results' section shows a table with one row:

Key	Assignee	Created	Issue Type	Priority	Reporter	Status	Summary	Updated
CALC-153		02/Sep/22 2:28 pm	Test	High	Sérgio Freire	TO DO	usability test	16/Aug/23 10:41 am

d.

2. Use the previous saved filter on the Test Plan configuration, as a regular Jira filter.

Learn more

- [JQL functions in Jira cloud](#)
- [JQL fields in Jira cloud](#)