Planning tests from a "methodology" perspective

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.

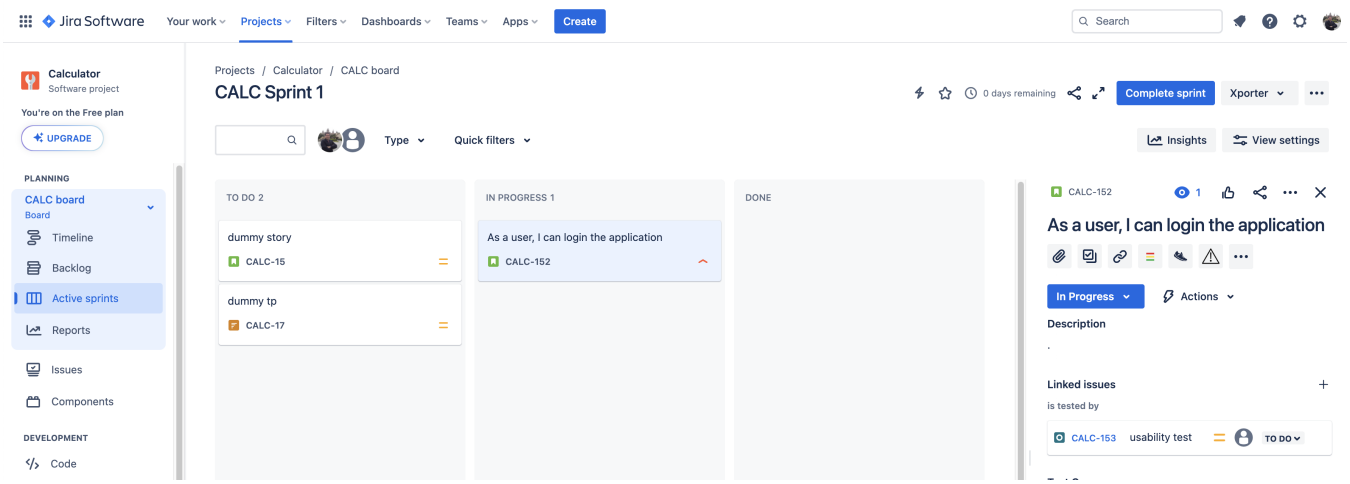
JQL features in Jira Cloud

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 advice to 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).

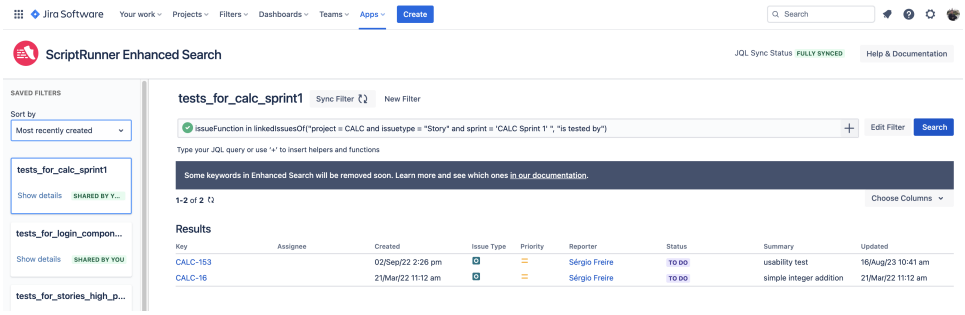
In this example, we have an active Sprint named "CALC Sprint 1" with 2 user stories, each one covered by some Test(s).



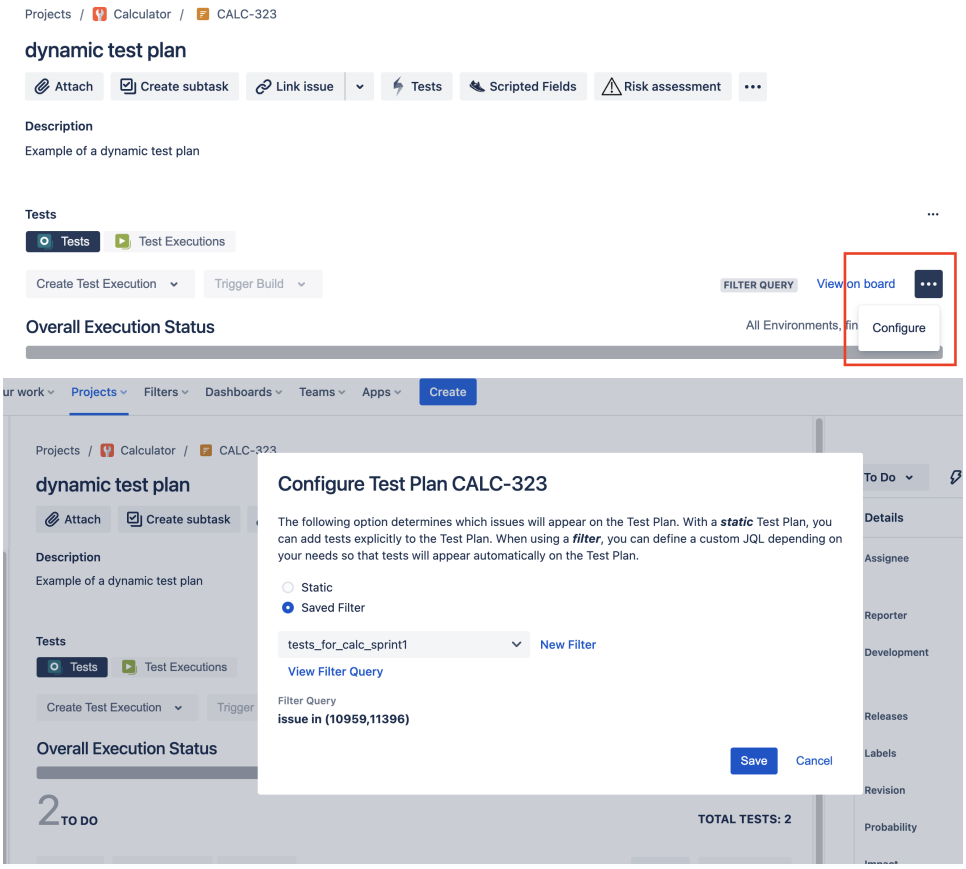
1. In **Apps** > **ScriptRunner Enhanced Search** create a JQL Filter named "tests_for_calc_sprint1" that lists all Tests covering user stories in the sprint named "CALC Sprint 1".

- **Sample JQL**

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

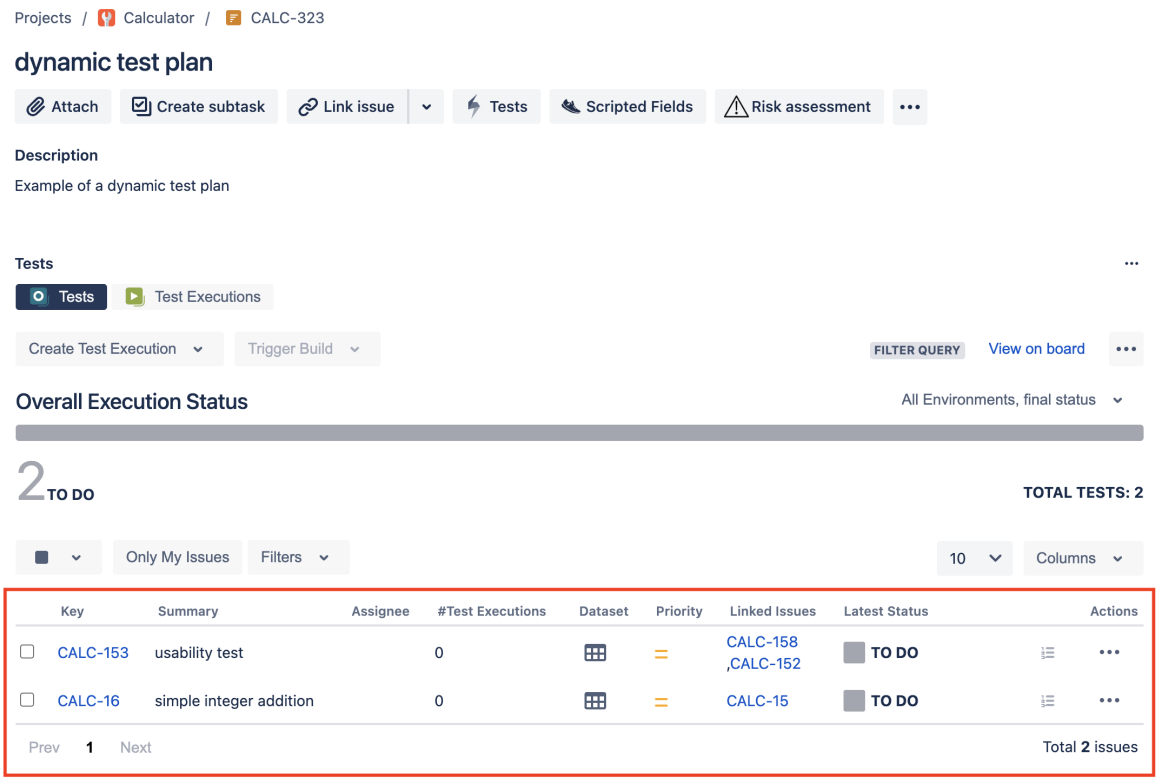


2. We then create a Dynamic Test Plan based on that query. To do so, we create a Test Plan as usual and then configure it use a saved filter.



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.



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

Context

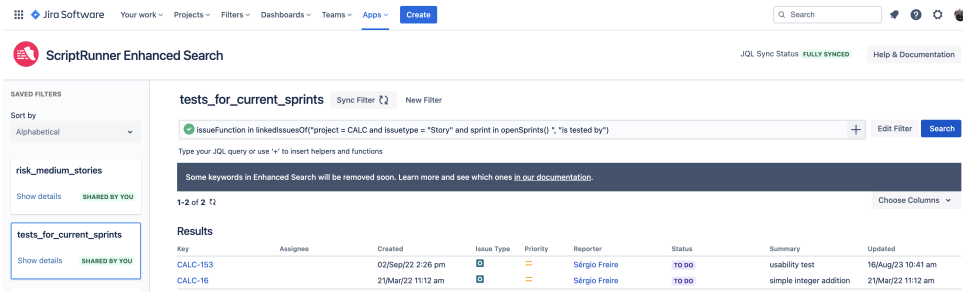
- A tester wants to create a Test Plan with all the tests that cover requirements in the current open/active sprint(s).

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).

1. 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")
```

    b. 

2. 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

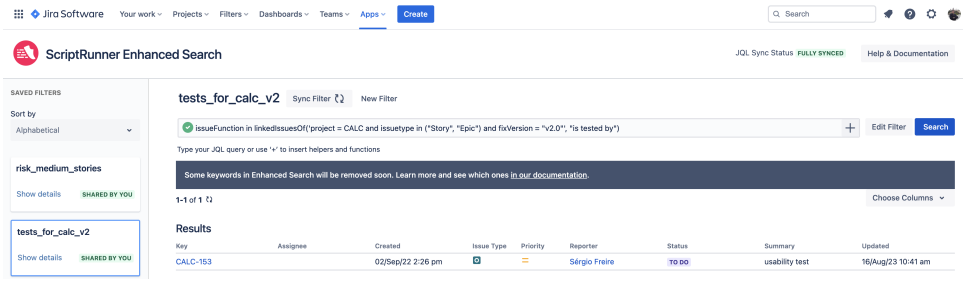
To be able to implement this use case, we'll use ScriptRunner Enhanced Search feature of the ScriptRunner app (you may also be able to implement this using another app with similar capabilities).

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. 

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.

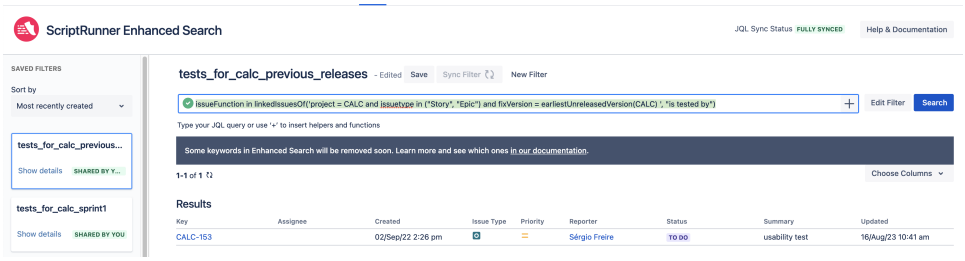
1. In **Apps** > **ScriptRunner Enhanced Search** create a filter (e.g., "tests_for_previous_releases") for the following JQL query.
    a. **Sample JQL**

```
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")
```

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