Using Xray in an Agile context

Executive Summary

Xray is used by many Agile teams, as well as by teams who are undergoing an Agile transformation and are moving out of traditional waterfall scenarios.

One of the primary points of being Agile means having a tool that adapts to your team's needs. This document sums up the different possibilities you have in working with Xray in an agile context and provides some guidance in getting started. Of course, we always suggest you evaluate what best fits your team's needs.

In order to have a successful Agile process, you need to have these processes/mindsets in place:

- **immediate information/no handoffs**: Xray provides immediate information visibility right within your work item. This way, you don't have to ask someone to know about results, testing progress, or what is the coverage status of an Epic/Story. For example, you can see how a user story is from a QA standpoint, right from the Story issue screen; likewise, you can see QA feedback in your existing Scrum Boards
- **collaboration**: everyone is invited to collaborate, no matter their role; therefore, anyone can interact and collaborate on user stories, test cases, test planning, test execution. This is because Xray works inside Jira and uses different issue types as an abstraction of its own entities, meaning you can collaborate inside Jira, the Jira way

In short, after successful installation & setup, we advise you to:

- create a Test Plan(s) per sprint
- configure your existing Agile Boards (i.e. Scrum Board of active sprint) to show Test Plans and (Sub) Test Executions assigned to your Sprints and to show the coverage status on the covered issues

However, as there is much more to know besides these points, please read the following sections to have a thorough understanding of how to take advantage of Xray in an Agile context.

Xray in Scrum based teams

If you're following Agile, particularly the Scrum framework, you'll have sprints of one, two, or more weeks (a sprint has a short, well-defined time period).

A sprint is a more manageable way of dividing the complexity of an entire release.

Each sprint represents one iteration that provides at its end a shippable product. Testing can occur hopefully throughout the whole sprint but sometimes it happens more closely to the end of it (something you should try to avoid).

Should you use Sub-Test Executions or not?

As Xray is a flexible tool, and every team has its own demands, we highly recommend you start simple and explore some of the possibilities offered by the tool.

Following the Agile principles, your processes should be reviewed often by the team and adapted accordingly with the team's needs.

You should also not dictate long and very strict processes, as you want to promote "individuals and interactions over processes and tools". Therefore, you need to find the right balance.

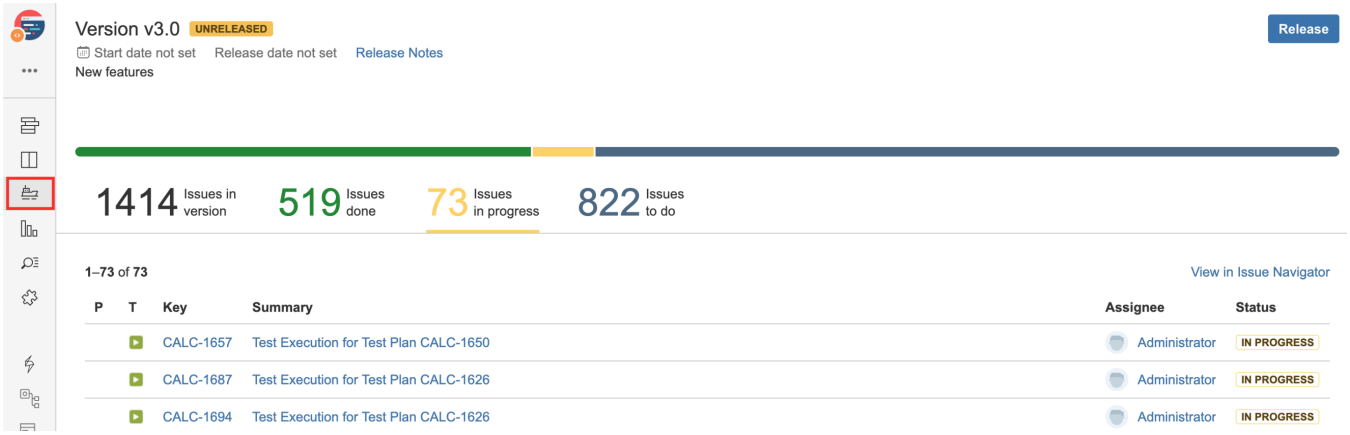
Process

- review the user stories with your team (remember: testers are part of the team)
- create Tests for each user story
- organize the Tests
- create one or more Test Plans (see above) and add the relevant Tests
- schedule multiple Test Executions, from the Test Plan or directly from each Story as Sub-Test Executions
- track overall, consolidated progress at the Test Plan level
- use reports to assess additional information, such as...
    - Overall Coverage Report, for evaluating the current coverage status of the coverable entities (e.g. user stories)
    - Traceability Report, to quickly filter out relevant entities based on their coverage, check the runs of related Tests and reported defects
    - Test Executions Report and/or Test Plans Report, as a means to see the results obtained on different environments, on different revisions of the SUT, and to check the reported defects if they're still to be resolved or not
- use Jira dashboards to share information about your project, including testing progress; you can use specific gadgets for this later

What belongs to a release?

Tests, Pre-Conditions and Test Sets are reusable entities; thus, in typical scenarios, they do not belong to the scope of a release or of a sprint.

On the other hand, Test Executions, Sub-Test Executions and Test Plans represent work targeted for a given release and/or a given sprint. Thus, they can be tracked as any other issue that needs to be handled in the scope of your release/sprint. If you have these issues assigned to some release through the FixVersion/s field, then they will appear in the summary information of the Releases project page as shown ahead.



Please note

As of Xray v3.4, Tests created from the "requirement"/story screen will inherit the FixVersion/s from the parent issue. The version assigned to the Test case has no special meaning unless you want to handle it as specification work that you need to address in the context of that release. If that's not the case, you can simply ignore.

(Xray may change this behaviour in the future.)

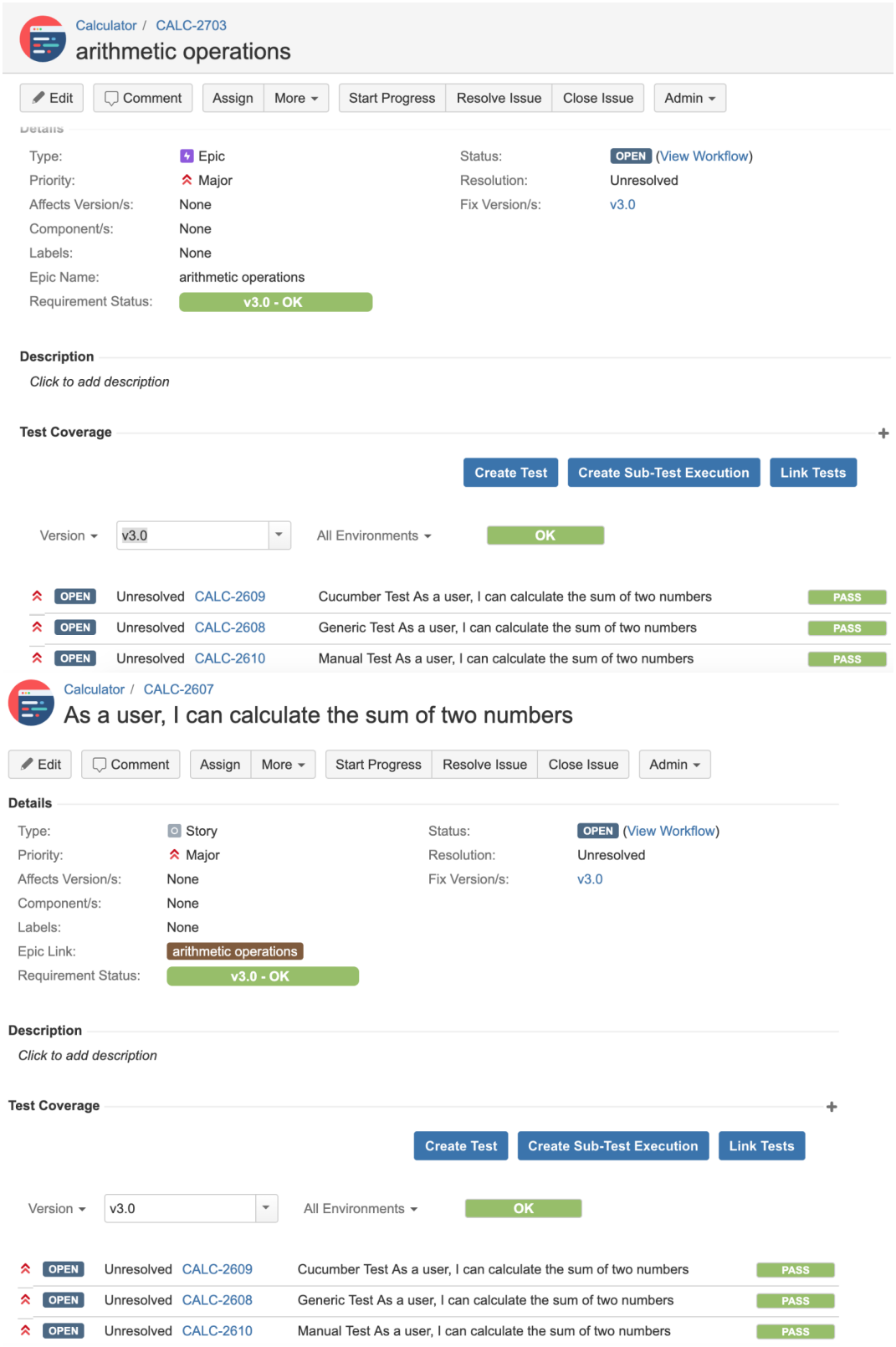
Coverage: Epics and Stories

Xray understands the "hierarchical" relation between Epic<=>Story. In fact, it's just one of the possible scenarios for handling parent and "sub-requirements".

That means that Tests covering a given Story will implicitly cover the related Epic. Thus, from the Epic screen, you can track its coverage, including the latest Test results based on the coverage of related Story issues.

Please note

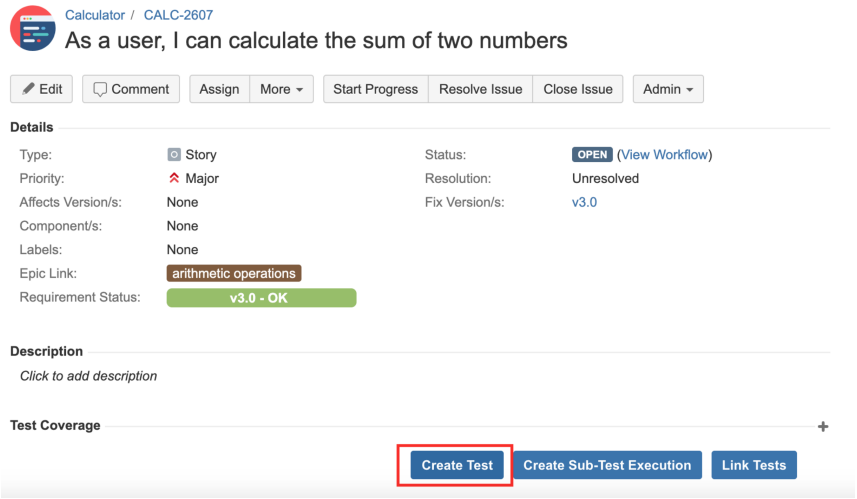
Please make sure you have the Epic<=>Story relation enabled under Xray "Issue Type Mappings" administration section.



Note that you can create and/or link a Test explicitly with an Epic, if you wish to cover it directly.

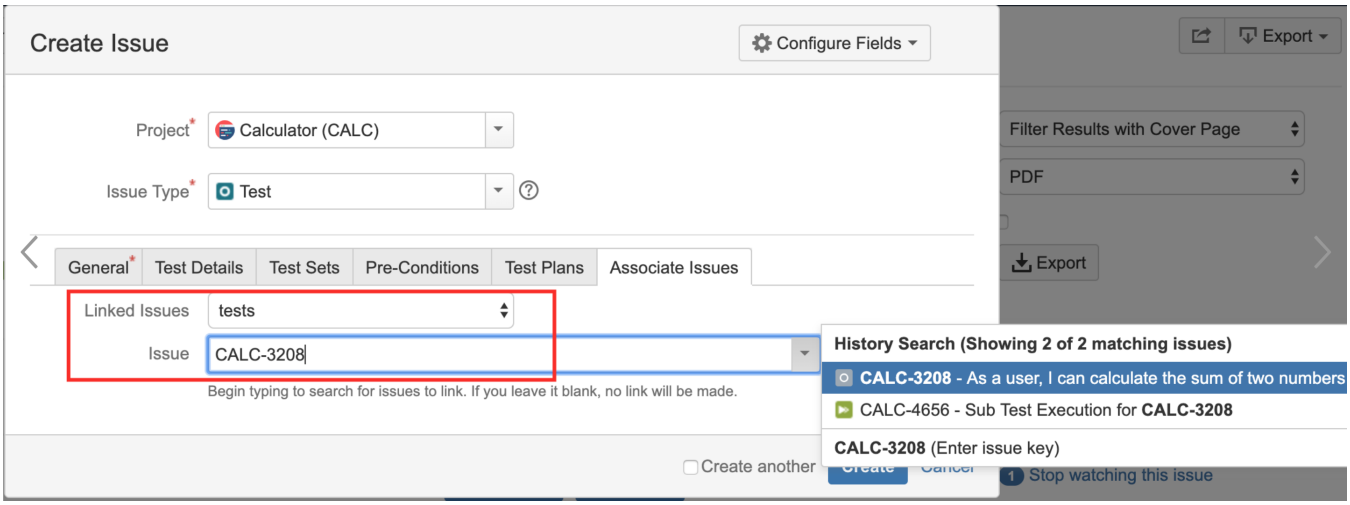
Creating Test cases

Tests can be created right from the Epic/Story screen. That way, they will be automatically linked with the respective issue.

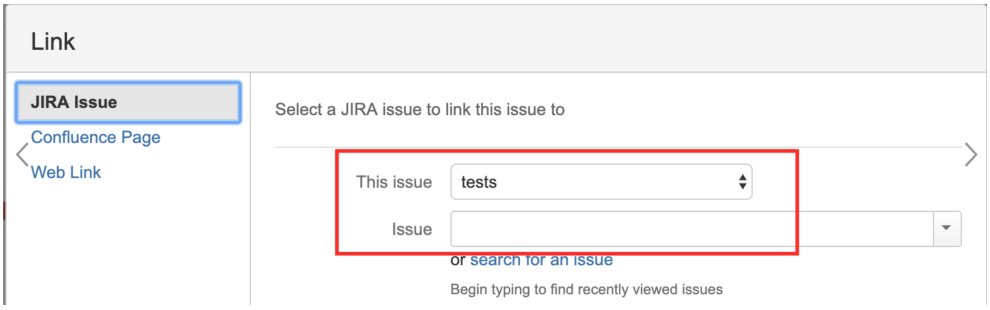


Otherwise, if you create Tests using the Create button from the top navigation bar, or from a Test Set or Test Repository folder action, then you have to specify the covered issue.

You can do that at the moment of creation...



or later on, by using the **More > Link** action available from the Test issue screen (or from the Story/Epic issue screen, but using the "tested by" issue link in that case).



Organizing the Test cases

Organization is essential to easily finding Tests later on. For example, whenever you need to perform regression testing.

Addressing the Tests directly related with some sprint (i.e. with the stories or any other entity tested and assigned to the scope of the sprint) is relatively easy: you just need to link the Tests to the stories. You don't need to organize the Tests in any special or structured way.

However, later on, you'll need to find the Tests by some sort of criteria:

- Tests associated with some high-level business case
- Tests associated with some component
- Tests having some sort of Priority
- Tests related with security or performance
- etc.

Thus, you'll need a structured way to organize your Test cases to make your work more efficient.

Possible organization scenarios

Xray provides several different options for organizing Test cases.

First, every Test that you create will be added as an issue of your project.

Thus, you can add fields and/or labels as a means to clearly categorize them, so that you can find them later on using standard issue search mechanisms, including JQL functions.

But there are more structured approaches to keep your test cases organization sane and manageable.

Overall, you have these organizational possibilities:

- **Implicit**, using labels and/or additional custom fields;
- **In flat lists called "Test Sets"**; a Test Set is implemented as an issue, thus you can add additional fields or labels and search for them using JQL;
- **Within folders and sub-folders**, inside the Test Repository of your project;
- **Using the Structure app**, if you need more complex organization scenarios (e.g. if you need to present Test cases and other entities in the same visual representation)

Which approach should you use?

There is no single answer that can suffice all sorts of team needs; the best is trying out the different approaches and choosing the one that makes your team more "comfortable" and effective. If your team loves JIRA and is keen on JQL, they'll probably prefer using flat lists (i.e. Test Sets). On the other hand, if your team has no deep Jira knowledge, is coming from legacy tools, or aim to manage thousands of Tests per project, then the hierarchical organization provided by the Test Repository may be the best approach.

In big enterprise scenarios following SAFe or similar frameworks for "Agile at Scale", the Structure app may cover more complex needs; please check the Test Repository approach first and go with Structure only if you need more advanced scenarios where you want to show Tests alongside other entities.

Learn more

Each organization approach provides its own benefits and its own limitations. Check out some comparison information providing better insights on the possibilities and limitations of some of these approaches

- Test Repository vs Test Sets
- Using Structure app vs Test Repository & Test Plan Board
- Tips for organizing Tests

Regression testing

The list of Tests you wish to run for regression testing may be quite dynamic and change over time, or it can be quite static; it all depends on your internal process.

If you wish to maintain an explicit and well-defined list of Tests to be used for regression testing, you can have a Test Set with those Tests. You can build this Test Set by searching Tests based on their attributes/fields, based on other existent Test Sets that group Tests by some criteria, or even based on some of the Test Repository folders. Maintaining this Test Set up-to-date may require some work.

On the other hand, you can have an implicit way of defining the Tests you wish to run during regression testing. How? Well, you may start by creating a first Test Plan for regression testing, adding all the Tests that you consider relevant. On upcoming releases, you may clone (using "issue clone" operation) the previous Test Plan and add/remove the Tests that you want.

Defining the scope of your testing using Test Plans

During the lifetime of a sprint, you will perform testing as a way to ensure you have a shippable product. Therefore, you'll need at least one Test Plan as a means to identify what you want to test and also to track its results.

The Test Plan can contain only Tests related with the "requirements" (user stories in this case) being addressed in the scope of the sprint but you can also use that Test Plan to track the testing related with the validation of previously implemented features (i.e. regression testing).

Thus, you can simply have one Test Plan per sprint. However, you may have more than one if you wish to manage and track independently different subsets of Tests.

From within each Test Plan you can schedule multiple "iterations" (i.e. Test Executions) for some or all of the Tests tracked by the Test Plan.

The Test Execution, which represents a task to run a bunch of tests against a certain version+revision of the project (e.g. SUT, AUT, HUT), can be assigned explicitly to the sprint or not. By assigning it to the sprint, you can manage it as an artifact/work that needs to be done in the scope of the sprint. Similar to other issues being addressed within the sprint, you can take advantage of the workflow status to track its progress, from a work point-of-view.

Purpose of Test Plans

A Test Plan defines the testing scope (i.e. the testing that you aim to perform in some context). Test planning can occur in the context of a release, in the context of a sprint or in the context of "testing cycle" (i.e. some timeframe that you allocate to perform testing).

The Test Plan is used as a basis to schedule Test Executions for all, or a subset of the Tests being tracked within it.

As you may have multiple "iterations" (i.e. scheduling of multiple runs for the Tests by creating multiple Test Executions), you'll need a way to see the consolidated progress for all those "iterations." The Test Plan provides this by showing you the latest status for each Test being tracked within the Test Plan.

How many Test Plans?

In the simplest scenario, you can have just one Test Plan with all the tests that you aim to perform (no matter their nature, neither their type nor if they're regression related or not).

Eventually, you may have multiple active Test Plans at the same time. The main reason would be if you want to manage & track test progress independently.

Although it'll be up to you to decide, you may want to have more than one Test Plan,

- if you want to manage each Test Plan independently
- if each Test Plan lifecycle is independent
- if you want to assign each plan to different teams
- if you want to clearly distinguish the results from different tests, because they, for example, bring different value (e.g. automated testing vs manual testing, regression testing vs non-regression testing, integration vs E2E tests, functional vs non-functional testing)

Coverage Analysis

You can analyze your coverage (i.e. how your "requirements"/user stories are) from the release perspective, and for that, the number of Test Plans you have is irrelevant; what matters is the Test Executions assigned to that release and the respective Test Runs.

If you want, you may eventually perform coverage analysis just by considering only the Tests and related results performed in the context of some given Test Plan, so you can assess what requirements are being "covered" by that Test Plan and what is their status if we just consider the results performed in the scope of that plan.

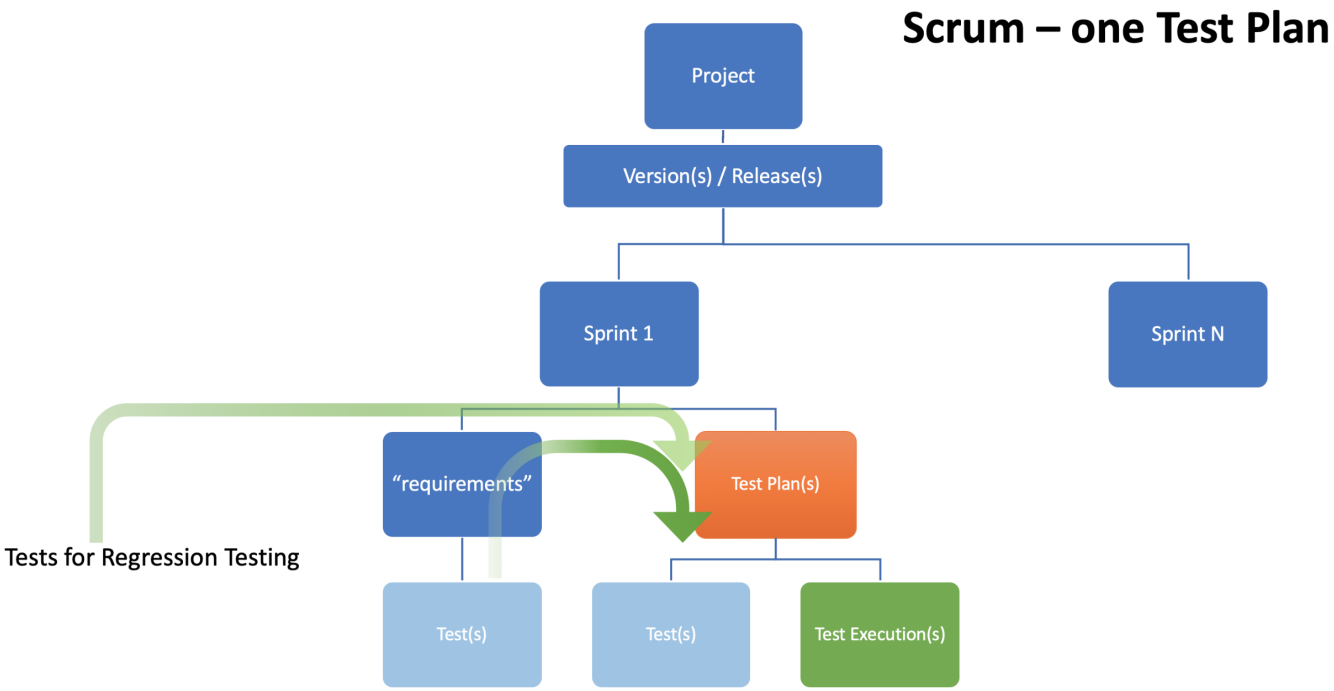
Basic: one Test Plan per Sprint

The simplest approach is to have just one Test Plan per sprint; you can set this explicitly by setting the Sprint field on the Test Plan issue.

By having the Test Plan being handled as an artifact to be handled in the scope of the sprint, similar to what happens with Stories, Tasks, Bugs and other issue types, you deal with it in the same way as you deal with other entities: it is something that you have to address/manage during the sprint. You can even take advantage of the workflow status of the Test Plan as a means to identify in what status it is (e.g. if it is in "To Do", "In Progress" or "Done").

If you're working in the context of some release, then you should also assign that Test Plan to release by setting its FixVersion/s field.

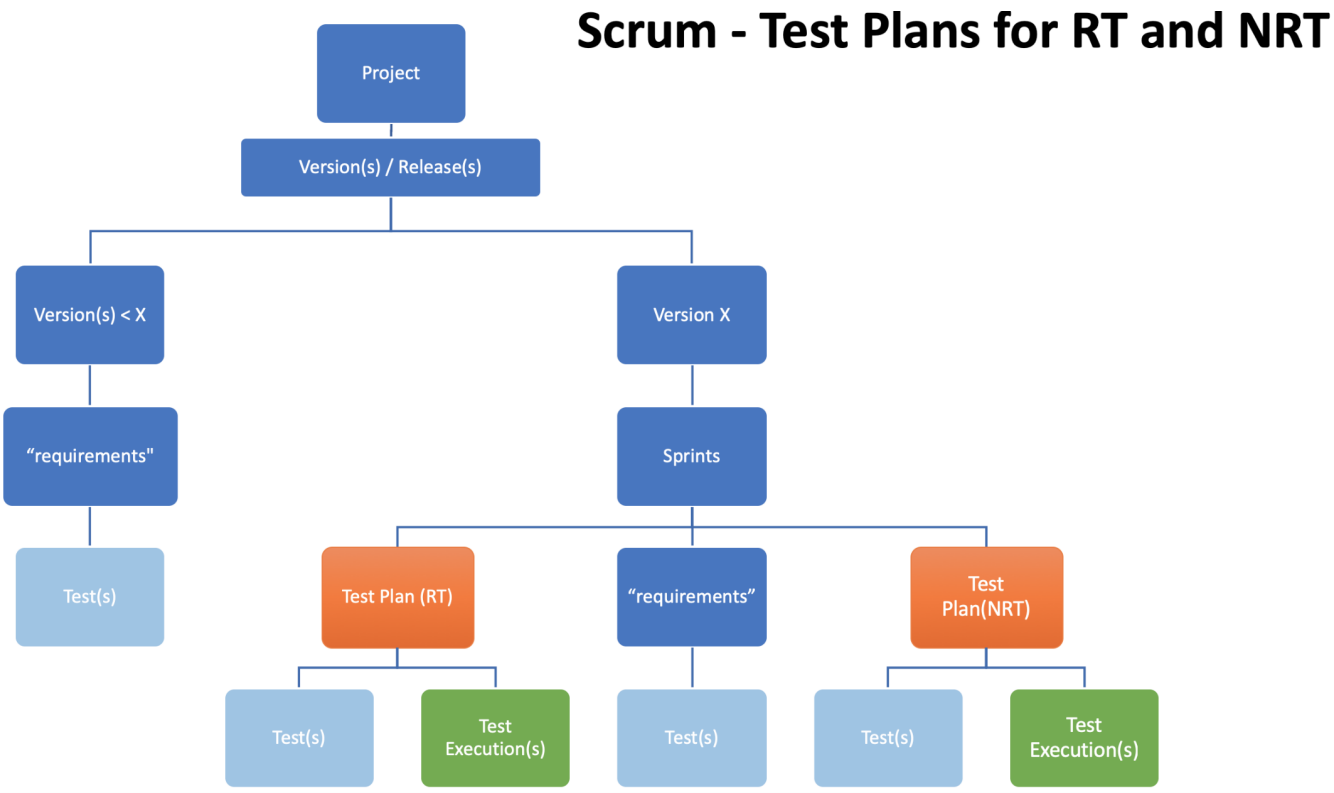
Test Executions created from the Test Plan will also, by default, inherit the Test Plan assigned version (i.e. FixVersion/s).



Test Plans for RT and NRT

Sometimes, you may prefer to distinguish regression testing (RT) from non-regression testing (NRT), for several reasons.

To do so, you can create one Test Plan containing all Tests that cover the "requirements" (i.e. user stories) and use it to track their progress along with another distinct Test Plan for all the regression related tests.



You can analyze your coverage (i.e. how your "requirements"/user stories are) from the release perspective or from each Test Plan perspective.

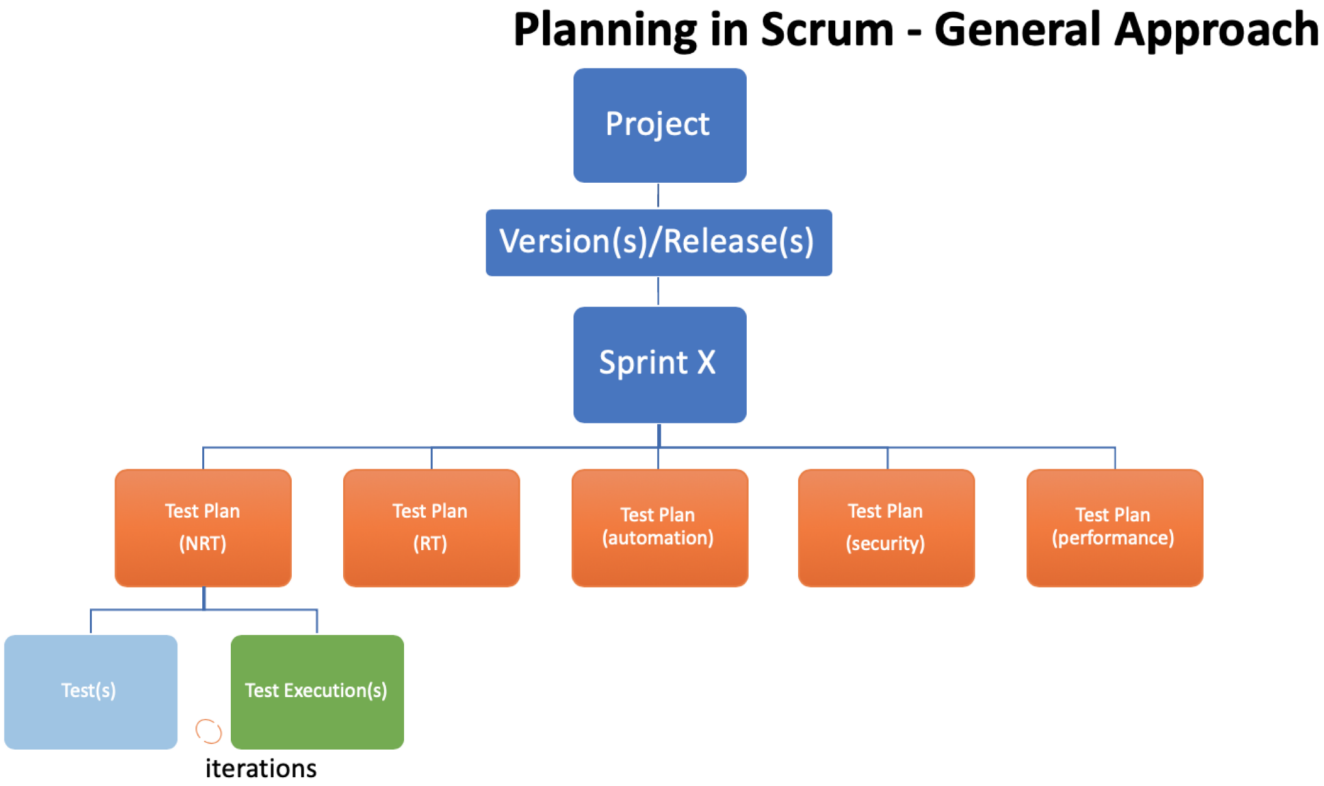
Multiple Test Plans

As a manager, you can define as many Test Plans as you wish.

Having multiple distinct Test Plans can give more clear insights about the testing progress of certain subsets of tests, eventually being managed by different users.

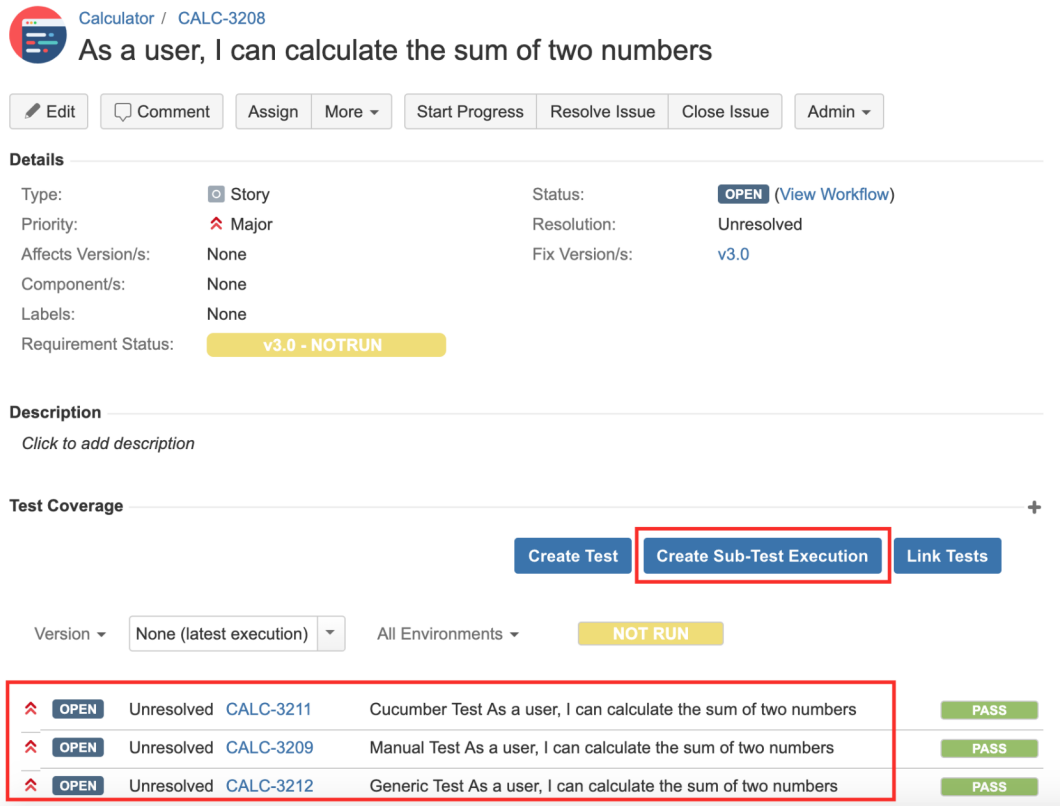
Of course, this may lead to some additional management overhead and thus you should choose wisely how many Test Plans you want to have.

The following diagram shows an example of possible different criteria for creating Test Plans.



Using Sub-Test Executions

Xray provides the ability to easily create Test Executions containing all the Tests that cover some "requirement"/Story as a sub-task of it.



This may provide a quick way for you to schedule the execution of certain Tests, and you can also link that Sub Test Execution to an existing Test Plan.

Sub-Test Executions mimic the creation of other development-related sub-tasks but in this case, for testing (execution only) concerns.

Overall, Sub-Test Executions provide:

- ability to track the progress of the Test Execution in the Agile board, by seeing it in context with the parent Story
- ability to count the time estimates of the Sub-Test Executions within the overall time estimate at the parent Story
- ability to optionally restrict workflow transition on the parent requirement based on the workflow status of the related sub-tasks (e.g. allow transition of the Story to "closed" only if the Sub-Test Executions are also "closed")

Using Sub-Test Executions may be a straightforward approach to define the "task(s)" of executing the Tests related to some Story. Thus, you can handle them similarly to any other development related sub-task in the context of the Story (e.g. performance analysis, visual design): assign them, estimate/log work on them, implement workflows on them to know whenever they're completed or need review.

However, if you have many Sub-Test Executions related to some Story, they may be hard to track.

Should you use Sub-Test Executions or not?

To be clear, you don't have to use Sub-Test Executions. However, as mentioned above, they may provide some interesting benefits for some specific use cases, including the ability to track them as ordinary sub-tasks of Stories.

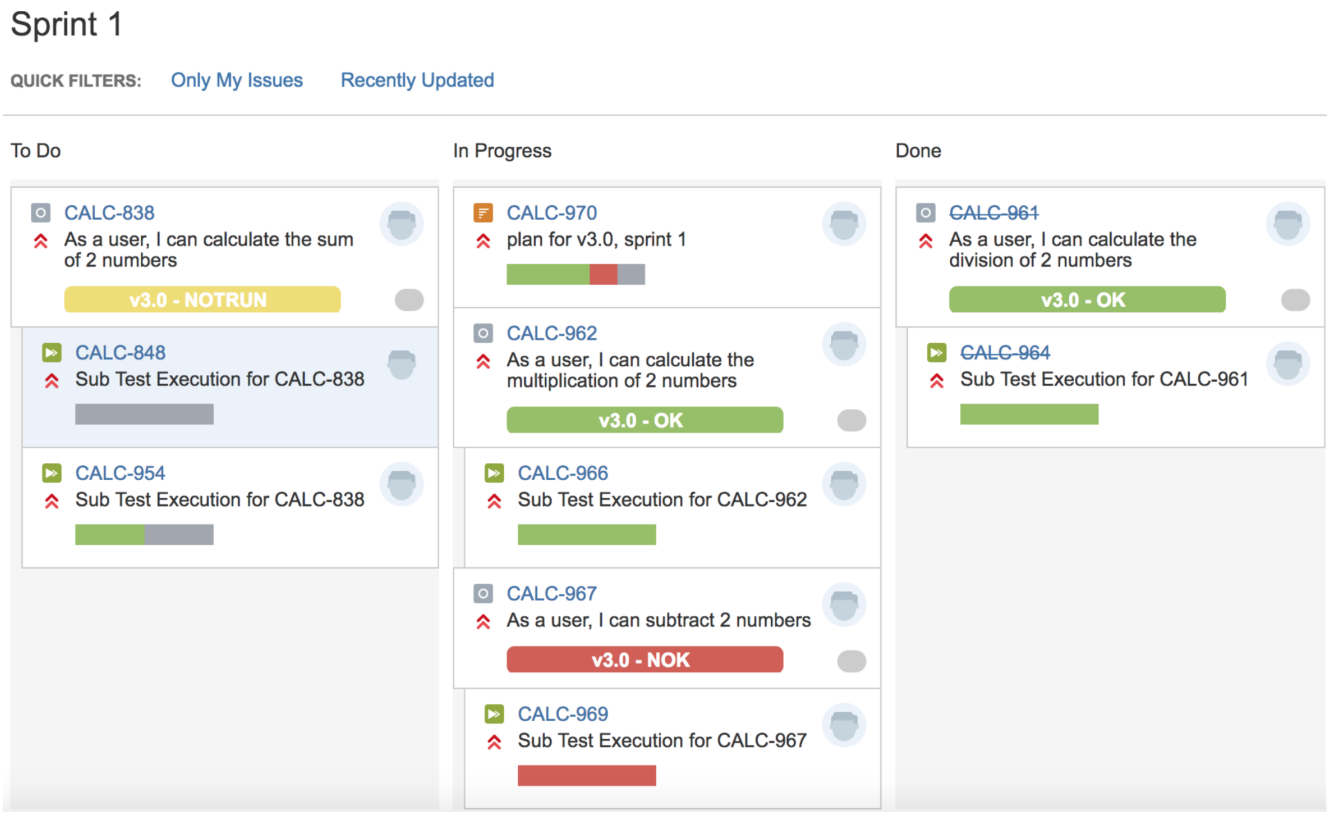
However, if you decide to also use Sub-Test Executions, please always link them to the proper Test Plan so you can track their results at a higher level: at the Test Plan level. That way, you will be able to always track the progress from the Test Plan point-of-view.

Tracking progress in Agile/Scrum based boards

As shown in the Agile Enhancements page, you can configure your Agile Scrum board to show immediate feedback about:

- **coverage status of the user stories** (or other entities that you cover with tests); note that this status includes real-time feedback about the latest test results
- **progress of Test Plans** (if you want, you may include Test Plans in the board)
- **progress of Test Executions** (if you want, you may include Test Executions in the board)

It's not recommended to Include in the board Test Executions coming from automated testing (i.e. during CI), as it will overload the board and not provide any real added value. Instead, you may include only Test Plan(s) as a means to group and show the consolidated results coming from automation.



Xray in Kanban based teams

You may have a look at the previous instructions for Scrum teams as most of it can also be applied to Kanban teams.

In Kanban, you're limiting the amount of WIP issues in a given state, namely, the ones that are in testing. Because you're not dividing your release into several periods of time, 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, namely, for tracking the regression testing. It may be a good 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 or not.

Questions

- We're not really adopting Scrum perfectly... can we use these approaches in our case?
  - Yes, you may follow these same principles and adapt them to your methodology.
- We have a hardening/clean-up sprint where we wish to test thoroughly, including deep regression testing. How can this be handled?
  - Hardening sprints are normally a sign that Agile is not fully being embraced. Having said that, you can have a Test Plan to track the testing performed in the scope of that sprint, adding all Tests related with user stories and other issues "implemented" on previous sprints, and thus consider the results performed in the scope of this Test Plans as being the relevant ones
- Are Test Plans, Test Executions or Sub-Test Executions child issues of an Agile board? In other words, do they belong to some Agile board?
  - No. All those issues belong to some project. They can optionally be assigned to a release (through the FixVersion/s field) and to a sprint (through the Sprint custom field).
  - A different thing is showing whatever issues you want in your existing boards; as those entities are implemented as issues, you can configure boards to show them. In fact, the same issues can be included (i.e. shown) in different boards.

Further reading

- Tips for planning tests
- How to implement QA in your projects
- Agile Board Enhancements