# How to implement QA in your projects

Having the Test Process in mind will help you implement quality assurance in your projects. The different testing phases (i.e., specifying, organizing, planning, executing, analyzing/reporting) are mostly implemented as issue types. More information about each phase can be obtained in each specific section within the User Guide.

> **Learn more**
>
> Please take some time to learn the terminology used in Xray and the relationships between its entities in Terms and Concepts.



## Managing versioned projects

In this use case, your project has one or more versions that you evolve as needed.

You may start with some requirements for v1.0 and later on create a v1.1 or a v2.0 release, and so on.

How do you then implement testing in this scenario?

### Mainly with manual testing

Suppose that you are working in version "XPTO" and you want to test it to make sure that the features you deliver are correct.

Your workflow would be more or less:

1. **Create "requirements"** (e.g., Story, Epic or other similar issue types) **and associate them with a version XPTO**, through the *FixVersion* field.
2. **Create one or more Tests to validate each requirement.** Typical manual Tests can be created from the requirement issue screen; thus, they will automatically be linked to the requirement. Cucumber automated tests can be created in the same manner, while other automated tests will be written in code and either linked to the requirement directly in the code or manually after importing their respective results.
3. **Organize your Tests either in lists (i.e., Test Set issues) or in folders**, so you can easily pick them afterwards whenever you need to create executions or plans. Test Sets can also be used as a way to indirectly validate requirements, since you can link them to requirements using the "tests" issue link.
4. **Create at least one Test Plan with the Tests you want to validate in version XPTO.** Don't forget to assign the Test Plan with version XPTO using the *FixVersion* field.

> **Learn more**
>
> Check out our Tips for planning tests, which explore the different testing possibilities, including in Waterfall and Agile methodologies.

5. **From the Test Plan, create one or more planned Test Executions with the Tests that you want to execute**. Each Test Execution is an abstraction of a "task for running some Tests" and can be assigned to specific users. Inside the Test Execution, invidividual Test Runs may be reasssigned to other users.
6. **Execute the Tests (i.e., Test Runs)**
   a. **For manual Tests, execute them in the scope of each TestExecution**. For each Test Run, report the status of each step or the overall result, if you prefer. You may need to create defects for failed Test Runs, which you can do immediately from a given step or globally at Test Run level.
7. **From the Test Plan, create new Test Executions to validate all Tests or just with the ones that are, for example, failing.**
8. **Use the prompt feedback of Test Plan and Test Execution issues along with reports to track the progress of your testing.** Built-in reports, such as the Traceability Report, Overall Requirment Coverage and others, along with custom dashboards can be used to track relevant information such as open defects.

### Mainly with automated testing

You may be implementing Continuous Integration and Continous Delivery with the help of automated testing. How can you adapt your process to this scenario?

If you're using an Agile methodology, such as Scrum, then you have Sprints that you can use as basis to define some scope.

Note that Scrum does not require you to make just one delivery at the end of each Sprint; you can deliver many during the lifespan of a Sprint.

Suppose that you are working in version "XPTO", sprint "X", and you want to test it to make sure that the features you deliver are correct.

Your workflow would be more or less:

1. **Create "requirements"** (e.g., Story, Epic or other similar issue types) **and associate them with version XPTO, through the *FixVersion* field, and sprint X.**
2. **Create one or more Tests for validating each requirement.** In this case, your automated tests will be specified before the actual implementation of the requirement is done, if you're following TDD, or after the requirement is implemented, in the worst case scenario. Cucumber automated tests can be specified in Jira (and implemented in code), while other automated tests will be written in code and either linked to the requirement directly in the code or manually after importing their respective results.
3. **Create at least one Test Plan with the Tests you want to validate in version XPTO.** Don't forget to assign the Test Plan with version XPTO and sprint X. Having a specific Test Plan for tracking the regression testing may prove to be useful.

4. **In the CI tool (e.g., Bamboo, Jenkins), run the automated tests and report them to Xray, associating them with the respective Test Plan**. In Xray, a Test Execution associated with the Test Plan will be created; it will contain the results for each automated Test. Test entities will be created automatically from the results, if they have not yet been created before.
5. **Analyze the results of each Test Execution**. For each failed Test Run, you may need to manually create defects, which you can do in the execution details screen of the respective Test Run.
6. **Use the prompt feedback of Test Plan and Test Execution issues along with reports to track the progress of your testing.** Built-in reports, such as the Traceability Report, Overall Requirment Coverage and others, along with custom dashboards can be used to track the relevant information such as open defects.

# Managing non-versioned projects

This may be common in Continuous Delivery scenarios or in the case where you simply don't want to manage versions at all. How do you implement testing in this scenario?

If you're using an Agile methodology, such as Scrum, then you have Sprints that you can use as basis to define some scope.

## Mainly with manual testing

Suppose that you are working in sprint "X" and you want to test it to make sure that the features you deliver are correct.

Your workflow would be more or less:

1. **Create "requirements"** (e.g., Story, Epic or other similar issue types) **and associate them with sprint X**
2. **Create one or more Tests for validating each requirement.** Typical manual Tests can be created from the requirement issue screen, thus they will be automatically linked to the requirement. Cucumber automated tests can be created in the same manner, while other automated tests will be written in code and either linked to the requirement directly in the code or manually after importing their respective results.
3. **Organize your Tests either in lists (i.e., Test Set issues) or in folders**, so you can easily pick them afterwards whenever you need to create executions or plans. Test Sets can also be used as a way to indirectly validate requirements, since you can link them to requirements using the "tests" issue link.
4. **Create at least one Test Plan with the Tests you want to validate in sprint X.** Don't forget to assign the Test Plan with sprint X.
5. **From the Test Plan, create one or more planned Test Executions with the Tests that you want to execute**. Each Test Execution is an abstraction of a "task for running some Tests" and can be assigned to specific users. Inside the Test Execution invidividual Test Runs may be reasssigned to other users.
6. **Execute the Tests (i.e. Test Runs)**
    a. **For manual Tests, execute them in the scope of each TestExecution**. For each Test Run, report the status of each step or the overall result, if you prefer. You may need to create defects for failed Test Runs, which you can do immediately from a given step or globally at Test Run level
7. **From the Test Plan, create new Test Executions to validate all Tests or just with the ones that are, for example, failing.**
8. **Use the prompt feedback of Test Plan and Test Execution issues along with reports to track the progress of your testing.** Built-in reports, such as the Traceability Report, Overall Requirment Coverage and others, along with custom dashboards can be used to track relevant information such as open defects.

## Mainly with automated testing

You may be implementing Continuous Integration and Continous Delivery with the help of automated testing. How can you adapt your process to this scenario?

If you're using an Agile methodology, such as Scrum, then you have Sprints that you can use as basis to define some scope.

Note that Scrum does not require you to make just one delivery at the end of each Sprint; you can deliver many during the lifespan of a Sprint.

Suppose that you are working in sprint "X" and you want to testing it to make sure that the features you deliver are correct.

Your workflow would be more or less:

1. **Create "requirements"** (e.g., Story, Epic or other similar issue types) **and associate them with sprint X.**
2. **Create one or more Tests for validating each requirement.** In this case, your automated tests will be specified before the actual implementation of the requirement is done, if you're following TDD, or after the requirement is implemented, in the worst case scenario. Cucumber automated tests can be specified in Jira (and implemented in code), while other automated tests will be written in code and either linked to the requirement directly in the code or manually after importing their respective results.
3. **Create at least one Test Plan with the Tests you want to validate in sprint X.** Don't forget to assign the Test Plan with sprint X. Having a specific Test Plan for tracking the regression testing may prove to be useful.
4. **In the CI tool (e.g., Bamboo, Jenkins), run the automated tests and report them to Xray, associating them with the respective Test Plan**. In Xray, a Test Execution associated with the Test Plan will be created; it will contain the results for each automated Test. Test entities will be created automatically from the results, if they have not yet been created before.
5. **Analyze the results of each Test Execution**. For each failed Test Run, you may need to manually create defects, which you can do in the execution details screen of the respective Test Run.
6. **Use the prompt feedback of Test Plan and Test Execution issues along with reports to track the progress of your testing.** Built-in reports, such as the Traceability Report, Overall Requirment Coverage and others, along with custom dashboards can be used to track relevant information such as open defects.