Quality Gates: Implementation examples

What you'll learn

Overview

Software quality gates are predefined checkpoints or criteria that must be met at various stages of the software development lifecycle to ensure that the software meets specific quality standards. Quality gates are used to assess the software product's quality, reliability, and readiness before proceeding to the next phase of development or deployment.

Jira (and Xray) can also be used to define Quality Gates (QGs) as in Jira we can define custom workflows or implement rules and conditions to define those QGs. Jira's ability to integrate with other tools such as CI/CD tools also makes it a good candidate for QGs.

To have valuable Quality Gates in place you must consider some key strategies, such as defining a clear criteria, the ability to integrate with automated tools, the possibility to establish gatekeepers and make sure to have a practice of continuous refinement.

Prerequisites

We will use the ScriptRunner tool for these examples, which is present when you install Xray in your Jira instance.

We will need:

- Access to ScriptRunner
- Permissions to alter workflows in Jira
- Permissions to create dashboards in Jira

Examples

In the below section we present possible implementations of Quality Gates using Jira and Xray data, in each example you will have information on how to define it and a final section on how you can follow the adherence using built-in reports from Xray.

We will showcase different possibilities of quality gates implementation focusing on data present in Jira by default and a combination of data present in Jira and Xray. There are a lot of possible quality gates that you can implement depending on what you want to follow, these are just two examples to showcase the possibilities using Jira and Xray.

Quality Gate 1 : User Stories can only be closed if all tests are successful

Description

User Stories can only be closed if all tests are successful.

Metric

The calculated overall status of a requirement based on the status of each test execution associated with it.

QG

Requirement Overall Status == OK or Requirement Overall Status != NOK

This QG is based on the "Requirement Status" custom field, which typically is configured to show the status of the covered item based on the latest results (no matter for which version).

To assess results for a given version we would need to use JQL, either by using the requirements() JQL function or using the search based on the "Requirement Status" custom field.

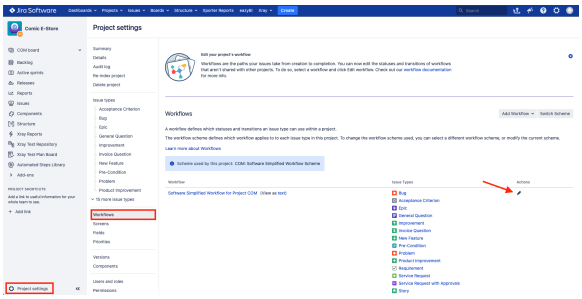
```
issue in requirements('OK','Calculator', 'v1.0')

Or
issuetype = 'New Feature' and "Requirement Status" = "v1.0 - OK"
```

Enhanced querying with JQL

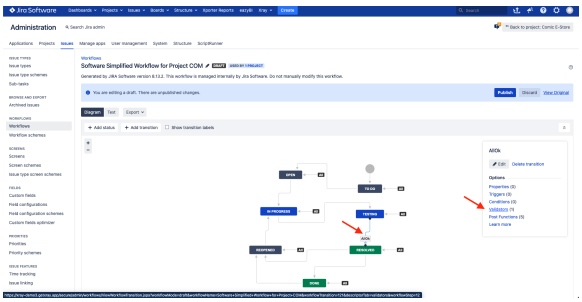
Implementation

In order to define that Quality Gate we are going to use the validators of workflows available in Jira combined with the custom field added by Xray in the requirements with the overall execution status of tests.

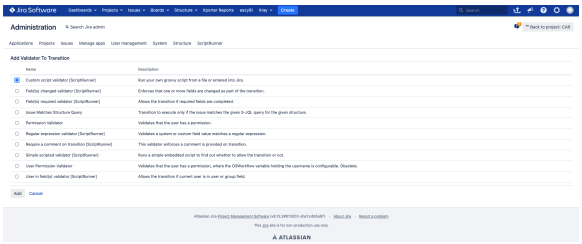
Go to *Project settings -> Workflows* and edit the workflow associated with your sprint.



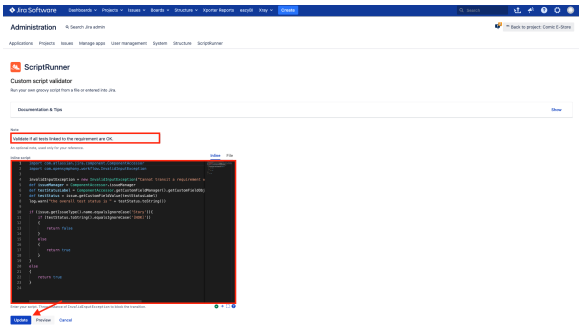
Choose the transition that you want to associate a validator with, in the example below it's the transition from Testing stage to Resolved stage, and press *Validators.*



In the next screen choose the validator type that you want to add, for this example we have chosen "*Cust om script validator [ScriptRunner]"* and press the *Add* button.



Fill in the Script and the optional note (useful information shown to the user when the transition is not allowed) and press Update.



The code used in the script validates that the requirement has the overall status of execution OK, meaning that it is covered by tests and the tests were executed successfully.

```
import com.atlassian.jira.component.ComponentAccessor
import com.opensymphony.workflow.InvalidInputException

def issueManager = ComponentAccessor.issueManager
def testStatusLabel = ComponentAccessor.getCustomFieldManager().
getCustomFieldObject("customfield_12500")
def testStatus = issue.getCustomFieldValue(testStatusLabel)
log.warn("the overall test status is " + testStatus.toString())

if (issue.getIssueType().name.equalsIgnoreCase('Story')){
   if (testStatus.toString().equalsIgnoreCase('[NOK]'))
   {
            invalidInputException = new InvalidInputException("Cannot
transit a requirement with failure in tests.")
        return false
   }
   else
   {
       return true
   }
}
else
{
   return true
}
```

> The validator will be active after updating the workflow.

Upon activation, any transition from the Testing stage to the Resolved stage will be unsuccessful if certain test executions fail.

Quality Gate 2 : Only successful Test Plans can be closed

Another example of a Quality Gate is to prevent closing Test Plans with failing test executions.

Description

Test Plans can only be closed if all tests associated to the test plan are passing.

Metric

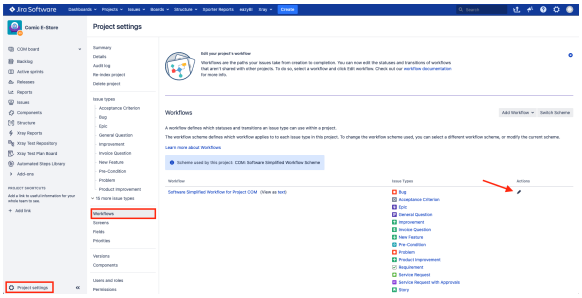
The passed percentage of tests associated to the test plan.

QG

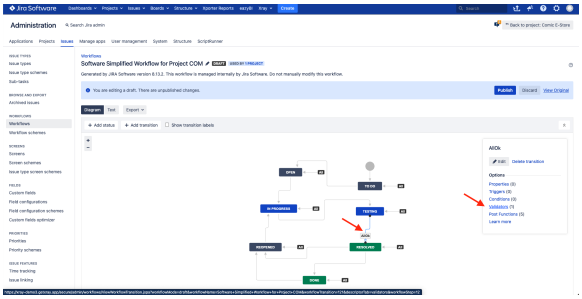
Passed percentage == 100%

Implementation

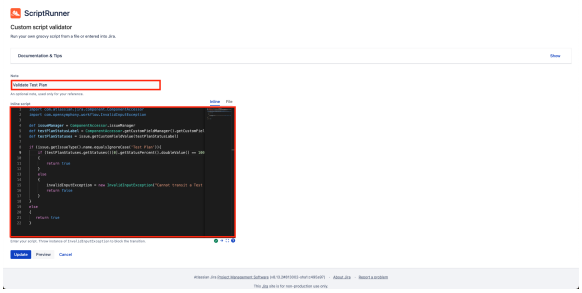
To add a validator that will prevent a Test Plan with unsuccessful test executions from being closed we must access "*Project Settings -> Workflows*" and edit the workflow by clicking the pencil icon below "*Actio ns*"



There we choose the transition that we want to add the Validator to by pressing on it, once we do it a pop up will appear where we can choose the Validators option.



We add a new Validator by adding a name to it and define a Jira Expression in the Script Validator window. We can also add a meaningful error message so that when the transition is not permitted users can understand why.

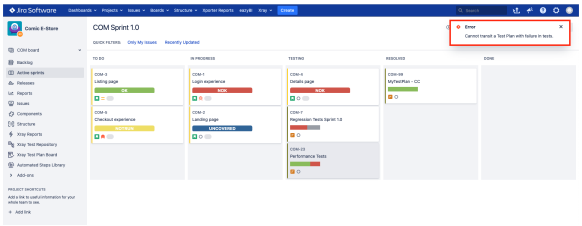


The Jira Expression used for this Validator is:

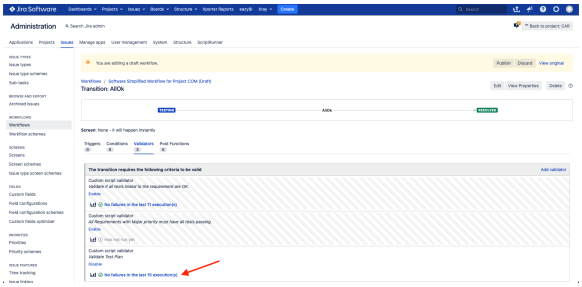
```
import com.atlassian.jira.component.ComponentAccessor
import com.opensymphony.workflow.InvalidInputException

def issueManager = ComponentAccessor.issueManager
def testPlanStatusLabel = ComponentAccessor.getCustomFieldManager().
getCustomFieldObject("customfield_11808")
def testPlanStatuses = issue.getCustomFieldValue(testPlanStatusLabel)

if (issue.getIssueType().name.equalsIgnoreCase('Test Plan')){
    if (testPlanStatuses.getStatuses()[0].getStatusPercent().doubleValue()
== 100.doubleValue())
    {
        return true
    }
    else
    {
        invalidInputException = new InvalidInputException("Cannot transit
a Test Plan with failure in tests.")
        return false
    }
}
else
{
   return true
}
```

This Jira Expression checks if the issue is a Test Plan and if the percentage of successful executions is 100%. If so the transition will be allowed otherwise we define an Exception with a meaningful message.

Notice that the message shown is the one defined above.



Note: To validate the defined rule you must update it and publish the workflow. Then go to your board and experiment by transitioning an issue, this will generate one entry in the audit logs available from the validators screen.



Track QGs with Jira and Xray

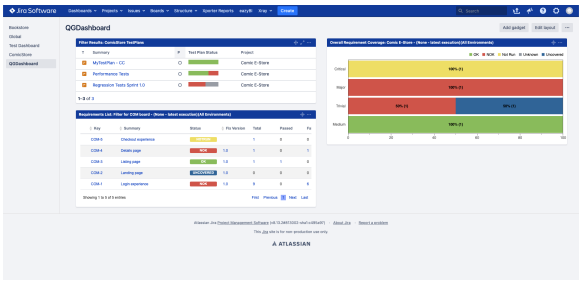
Jira Server/DC with Xray has a series of possible reports available out of the box, those reports are also available in the form of dashboard gadgets that allow users to define custom dashboards.

From those, two reports allow you to assess whether the Requirements are or are not being covered by tests:

- Overall Test Coverage (report that shows the Test coverage filtered by priority and the overall calculated status of the tests that are covering the requirement)

- Test Plan Report (report that shows a list of Test Plans with consolidated information for each one, including the test statuses count, overall progress and Test Environment-related metrics.)

Once added to a dashboard, we can check if we have requirements with unsuccessful tests and also what the status of the Test Plans is.



Note: You can customize these reports further by using JQL filters or using different options.

Tips

- Ensure that you define an informative error message in the Exception because it is the only information the user will see if a transition fails.
- Use <JIRA_ENDPOINT>rest/api/2/issue/<ISSUE_KEY>?expand=names to verify what fields are exposed and their ids.
- Use the audit logs to view the validators execution and output.