

Performing Risk-Based Testing (RBT) with Xray

- [Overview](#)
- [Performing RBT with Xray](#)
 - [Where to set the risk](#)
 - [RBT related activities](#)
 - [Defining impact, probability and risk level](#)
 - [Adding Tests to a Test Execution/Test Plan/Test Set, sorted by risk level](#)
 - [Risk defined at Test level](#)
 - [Risk defined at requirement or at project level](#)
 - [Sort and re-rank Tests in a Test Execution by risk level](#)
 - [Risk defined at Test level](#)
 - [Risk defined at requirement or at project level](#)
 - [Sort and re-rank Tests in a Test Plan/Test Set by risk level](#)
 - [Analyzing overall requirement coverage by risk level](#)
 - [Risk defined at requirement or at project level](#)
 - [Overall Requirement Coverage Report](#)
 - [Overall Requirement Coverage gadget](#)
 - [Requirements List gadget](#)
 - [Analyzing testing results by risk level](#)
 - [Consolidated results](#)
 - [Overall Test Results gadget](#)
 - [Tests List gadget](#)
 - [Tests report \(legacy report on project tab\)](#)
 - [Finding out defects related with certain risk levels](#)
 - [Track and/or disallow changes on "requirements" and in test cases](#)
 - [Use different Test Plans based on Risk level](#)
 - [Setup and Configuration](#)
 - [Configuring saved filters](#)
 - [Risk at Test level](#)
 - [Risk at requirement or at project level](#)
 - [Configuring Risk, Probability and Impact](#)
 - [Using Jira natively](#)
 - [Configuration examples](#)
 - [Using Risk Management for Jira app](#)
 - [Configuring fields](#)
 - [Using it](#)
 - [Using Risk Register app](#)
 - [Configuring it](#)
 - [Using it](#)
 - [Using Jira Misc Custom Fields app](#)
 - [Calculating risk at Test level and storing it in a Calculated Single-select field](#)
 - [Calculating risk and storing it in a Calculated Number field](#)
 - [Calculating risk at requirement/project level and storing it in a Calculated Single-select field at Test level](#)
 - [Using ScriptRunner](#)
 - [Calculating risk at Test level and storing it in a Single Select field using a Script Listener](#)
 - [Calculating risk at requirement or project level and storing it in a Single Select field at Test level using a Script Listener](#)
 - [Other non-standard configuration approaches](#)
- [References](#)

Overview

This article provides background information on [Risk-Based Testing \(RBT\)](#) and its foundations on [Risk Management concepts](#), which will be briefly presented and clarified.

RBT is an approach used by testing teams whose focus is on assessing risks and how to handle them, from a testing perspective.

Since Xray v3.4, you may take advantage of some built-in capabilities in order to leverage RBT in your teams.

Performing RBT with Xray

Xray can be used to perform Risk-Based Testing, giving you all the flexibility to tailor it to your team's needs and implement RBT successfully.

Xray is not dependent on any risk management app nor does it provide in itself risk management capabilities. In other words, it's up to you how to define risk and how to manage it.

The first step is to decide [where to set the risk](#) (i.e. on requirements, Tests or at project level); that will affect how you will implement the related fields and how you will use Xray during RBT.

For risk management you need to choose if you're going to use a specific app (e.g. Risk Management for Jira, Risk Register) or not; using Jira or a generic customization app (e.g. Jira Misc Custom Fields, ScriptRunner, etc.) can be enough for risk identification and analysis.

After you have decided where to set the risk and how, and [configured](#) properly, you may perform RBT and use Xray to assist you.



Learn more

If you're new to Risk Management and RBT, please check the [Risk Management](#) article.

Where to set the risk

As long as you are able to define a sort-able value for the risk as a custom field in either cover-able issues (e.g. "requirements"/stories) or Test themselves, then you will be able to select and rank Tests for execution, based on risk level. If you have identified global risks at project level as issues, and those risks can be mitigated somehow by testing, then you can also use RBT with them.

Thus, you may define risks (and the corresponding risk level as a sort-able custom field on the related issue) at:

- **Test level**
- **requirement/story level**, or any other issue type that can be covered with test cases (i.e. configured as "requirement issue types" on "[Issue Type Mapping](#)" settings).
- **project level**, using a specific issue type

Each approach has its own [usage scenarios and pros/cons](#). Throughout this article, most of the examples will be based on risks defined at Test level.

Depending on your approach to Risk Management and to RBT, you will need to configure risk related fields as explained in the [configuration section](#) ahead.

RBT related activities

The whole [RBT process](#) has many activities and Xray can be used along the way.

This section provides a brief overview of some of them giving special focus on sorting and ranking Tests using the risk level as input; however, there's a lot more than that.

Defining impact, probability and risk level

For a given identified risk, during Risk Analysis you need to define impact and probability, as a means to calculate the risk level.

Thus, impact and probability are mostly temporary values used to calculate the risk; they can be assessed early and updated if needed during Risk Monitoring and Reviewing.

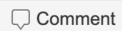
Where and how you define impact and probability fields depends on your approach, including how your fields will be implemented.

The following screenshots show risk being defined at different levels, while also using different tools to implement them.



Calculator / CALC-1203

CanDoStuff



Assign

More ▾

Start Progress

Resolve Issue

Close Issue

Admin ▾

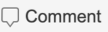
Details

Type:	<input checked="" type="radio"/> Test	Status:	OPEN (View Workflow)
Priority:	<input checked="" type="radio"/> Blocker	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	core		
Test Repository Path:	/other_tests		
Impact:	4		
Probability:	<input type="text" value="1"/>		
Risk Level:	L2: Medium		
Details:	4		



Calculator / CALC-4645

story1



Assign

More ▾

Start Progress

Resolve Issue

Close Issue

Admin ▾

Details

Type:	<input checked="" type="radio"/> Story	Status:	OPEN (View Workflow)
Priority:	<input checked="" type="radio"/> Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	v3.0 - OK		
JMCF - Risk Level:	L5: Severe		
Probability:	<input type="text" value="4"/>		
Impact:	4		



Calculator / CALC-4665

replacing relational database with a NoSQL one for authentication



Assign

More ▾

Start Progress

Resolve Issue

Close Issue

Admin ▾

Details

Type:	<input checked="" type="radio"/> Risk	Status:	OPEN (View Workflow)
Priority:	<input checked="" type="radio"/> Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

> Description

> Test Coverage

Risk assessment

☐ Specify Residual Exposure

Risk register

Exposure

HIGH

RR: Probability: Almost certain

RR: Impact: Major

Adding Tests to a Test Execution/Test Plan/Test Set, sorted by risk level

Risk defined at Test level

Add Tests to Test Execution CALC-3702

AssigneeIssue Assignee

SelectSearchJQL

Filter(s)

Project

Calculator (CALC)

Test Type

Choose the Test Type

Contains text

Risk Level

L1: LowL2: MediumL3: HighL4: Very HighL5: Severe

Risk Level Name (calculated from Risk CF)

Issue Type	Key	Summary	Priority	Impact
	CALC-3703	ability to perform clear / start new operations from scratch		4
	CALC-1202	CanAddNumbers		4
	CALC-1203	CanDoStuff		4
	CALC-3706	check the history of operations		2

Showing 1 to 4 of 4 entries

Columns

Search

Restore Defaults

Impact

Issue Type

Key

Priority

Probability

Risk Level

Summary

Affects Version/s

Asset

DoneCancel

ClearSearch

Hide Tests in non-executable Statuses

Add selectedAdd all (4)Cancel

Whenever adding Tests to an existing Test Execution/Test Plan/Test Set, you may choose to show the impact, probability and risk level related fields.

Tests can be ranked on descending order of risk level, so the ones related with higher risks can be addressed first; you may also sort Tests by impact or probability if you want.

To sort a column (i.e custom field) by descending/ascending order, just click on the column name.

	Issue Type	Key	Summary	Priority	Impact	Probability	Risk Level
		CALC-3703	ability to perform clear / start new operations from scratch		4	4	L5: Severe
		CALC-1202	CanAddNumbers		4	2	L3: High
		CALC-1203	CanDoStuff		4	1	L2: Medium
		CALC-3706	check the history of operations		2	1	L1: Low

Showing 1 to 4 of 4 entries

FirstPrevious1NextLast

On the left side, you may use filters on the impact, probability and risk level fields. Filtering by risk level is essential but filtering by other related files may prove to be useful.

Add Tests to Test Execution CALC-3702

Select
Search
JQL

Filter(s)

Project
Calculator (CALC)

Test Type

Choose the Test Type

Contains text

Risk Level
L1: Low
L2: Medium
L3: High
L4: Very High
L5: Severe

Risk Level Name (calculated from Risk CF)

Impact

Impact

Probability

Probability (RBT)

Search

Clear selected items

☒ Impact
☒ Probability
☒ Risk Level

Fields
☐ Summary
☐ Status
☐ Priority
☐ Resolution

Summary

C-3703
ability to perfo

C-1202
CanAddNumt

C-1203
CanDoStuff

C-3706
check the hist

Clear
Search

The way filters are shown depends on their type (i.e. the type of the custom field); therefore, it's important to setup these fields properly.

Risk defined at requirement or at project level

If risk is defined at requirement or project level then we need to first sort these entities by risk level; only then we may obtain related Tests based on the previous sorting.

How to perform this exactly?

- use saved filters to group requirements with similar risk level
 - create several issue filters, each one grouping requirements (or project level risks) having the same risk exposure. Using a nomenclature for the saved filters can help you out (example: "risks_low", "risks_medium", "risks_high")
 - obtain the related tests using JQL. Example: `issue in requirementTests("risks_high")`
 - add tests, group by group, by descending order of risk; whenever adding tests you may sort them by an additional criteria (e.g. Priority), including risk explicitly associated to it
- or use a calculated field on Test issues to perform the risk calculation using the risk level(s) of the associated requirement(s). By having this field "replicated" on the tests themselves, will ease handling of test cases during RBT as it will be similar to handling "Risk defined at Test level." Please see ahead a [configuration example](#) using ScriptRunner app.

Sort and re-rank Tests in a Test Execution by risk level

Risk defined at Test level

A Test Execution contains a list of tests to be run accordingly with their ranking (i.e. which you may see in the first column, named "Rank").

Tests

+ Add

Votes:
Watchers:

Overall Execution Status

5 TODO

TOTAL TESTS: 5

FILTERS

Test Set	Assignee	Status	Component	Search
All	All			Contains text ✕ Clear

Dates

Created:
Updated:
Begin Date:
End Date:

Agile

[View on Board](#)

> Environments

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Risk Level	Status
1	CALC-3706	check the history of operations	Manual	0	0	Administrator	L1: Low	TODO
2	CALC-3703	ability to perform clear / start new operations from scratch	Manual	0	0	Administrator	L5: Severe	TODO
3	CALC-1202	CanAddNumbers	Generic	1	0	Administrator	L3: High	TODO
4	CALC-3700	sample test	Manual	0	0	Administrator	L3: High	TODO
5	CALC-1203	CanDoStuff	Generic	0	0	Administrator	L2: Medium	TODO

Showing 1 to 5 of 5 entries

First Previous 1 Next Last

Tests may be visually ordered ascending/descending by clicking on the column name; this won't affect their ranking, which is the one used as the effective order to run the tests.

Tests

+ Add

Votes:
Watchers:

Overall Execution Status

5 TODO

TOTAL TESTS: 5

FILTERS

Test Set	Assignee	Status	Component	Search
All	All			Contains text ✕ Clear

Dates

Created:
Updated:
Begin Date:
End Date:

Agile

[View on Board](#)

> Environments

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Risk Level	Status
2	CALC-3703	ability to perform clear / start new operations from scratch	Manual	0	0	Administrator	L5: Severe	TODO
3	CALC-1202	CanAddNumbers	Generic	1	0	Administrator	L3: High	TODO
4	CALC-3700	sample test	Manual	0	0	Administrator	L3: High	TODO
5	CALC-1203	CanDoStuff	Generic	0	0	Administrator	L2: Medium	TODO
1	CALC-3706	check the history of operations	Manual	0	0	Administrator	L1: Low	TODO

Showing 1 to 5 of 5 entries

First Previous 1 Next Last

If you want to make this order permanent, by setting the rank accordingly, then you can choose "Apply Rank."

The screenshot shows a 'Test Ranking' dialog box with the question 'Apply current table order to Tests?' and 'Yes' and 'Cancel' buttons. Below the dialog is a table of test sets. The table has columns for Rank, Key, Summary, Test Type, #Req, #Def, Assignee, Risk Level, and Status. The tests are ranked 1 to 5 from bottom to top.

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Risk Level	Status
2	CALC-3703	ability to perform clear / start new operations from scratch	Manual	0	0	Administrator	L5: Severe	TODO
3	CALC-1202	CanAddNumbers	Generic	1	0	Administrator	L3: High	TODO
4	CALC-3700	sample test	Manual	0	0	Administrator	L3: High	TODO
5	CALC-1203	CanDoStuff	Generic	0	0	Administrator	L2: Medium	TODO
1	CALC-3706	check the history of operations	Manual	0	0	Administrator	L1: Low	TODO

TOTAL TESTS: 5

FILTERS

Test Set	Assignee	Status	Component	Search
All	All			Contains text <input type="text"/> <input type="button" value="Clear"/>

Show 10 entries Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Risk Level	Status
1	CALC-3703	ability to perform clear / start new operations from scratch	Manual	0	0	Administrator	L5: Severe	TODO
2	CALC-1202	CanAddNumbers	Generic	1	0	Administrator	L3: High	TODO
3	CALC-3700	sample test	Manual	0	0	Administrator	L3: High	TODO
4	CALC-1203	CanDoStuff	Generic	0	0	Administrator	L2: Medium	TODO
5	CALC-3706	check the history of operations	Manual	0	0	Administrator	L1: Low	TODO

Showing 1 to 5 of 5 entries

Note that you may also sort and re-rank Tests by impact or probability, if you want.

Risk defined at requirement or at project level

If you have a custom field at Test level that "inherits" (e.g. copies or computes) the risk level from the parent issue (e.g. requirement/story or project level risk issue) then you can follow the previous instructions described for risk defined at Test level.

Otherwise, there is not yet a straightforward solution.

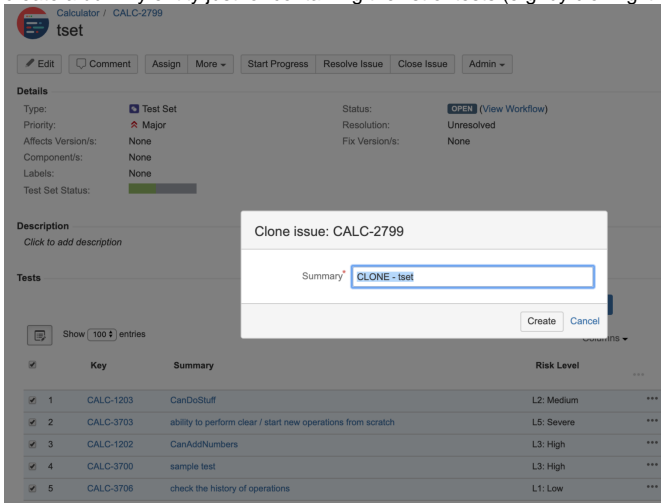
Sort and re-rank Tests in a Test Plan/Test Set by risk level

As of v3.4, Xray does not yet provide a way to perform sort and re-ranking on Test Plans or Test Sets based on specific custom field.

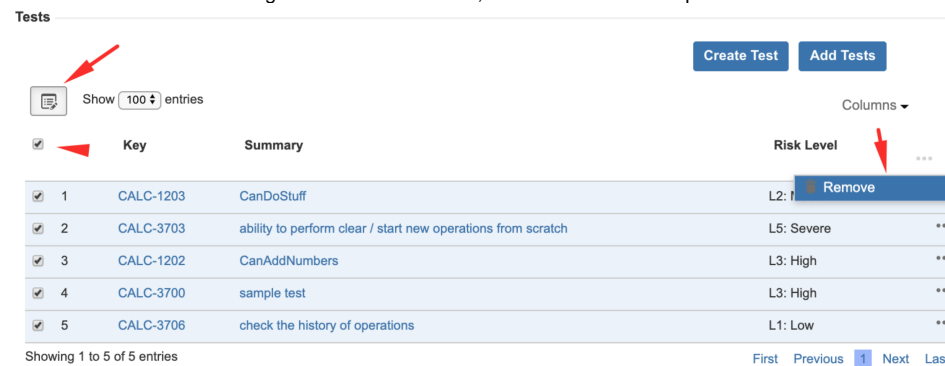
However, if you add Tests to an existing Test Plan/Test Set and you order them by risk level in the dialog before they're actually added, the Tests ranking in the destination entity will take into account that order.

There, one workaround/"hack", for now, would be:

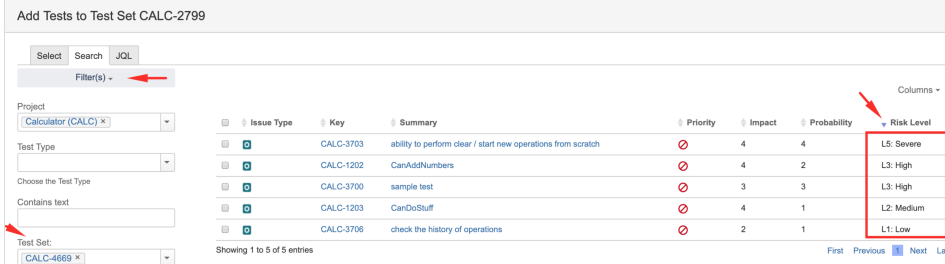
1. create a dummy entity just for containing the list of tests (e.g. by cloning it or by creating a dummy Test Set with those Tests)




2. remove the Tests from the original Test Plan/Test Set, as shown in this example for a Test Set




3. search the Tests, using JQL or filters to obtain the previously saved list of Tests from the temporary entity and sort the Tests by Risk Level



4. add Tests back to the original Test Plan/Test Set

 **Calculator** / CALC-2799

 **tset**

Edit

Comment

Assign

More

Start Progress

Resolve Issue

Close Issue

Admin

Details

Type: **Test Set**

Status: **OPEN** (View Workflow)

Priority: **Major**

Resolution: **Unresolved**

Affects Version/s: **None**

Fix Version/s: **None**

Component/s: **None**

Labels: **None**

Test Set Status: **No tests**

Description

Click to add description

Tests

Create Test

Add Tests

Show 100 entries

Columns

	Key	Summary	Risk Level	
1	CALC-3703	ability to perform clear / start new operations from scratch	L5: Severe	***
2	CALC-1202	CanAddNumbers	L3: High	***
3	CALC-3700	sample test	L3: High	***
4	CALC-1203	CanDoStuff	L2: Medium	***
5	CALC-3706	check the history of operations	L1: Low	***

Showing 1 to 5 of 5 entries

First

Previous

1

Next

Last

These instructions assume that Risk is defined at Test level; if it's defined at requirement or project level, then it would need to be adapted accordingly depending on your implementation.

Analyzing overall requirement coverage by risk level

Risk defined at requirement or at project level

If you have assessed risks at requirement or project level, then Xray offers you the possibility of analyzing coverage by risk level (and even by impact and probability). This is supported as long as those fields are "Select List (single choice)" based; number or text based custom fields are not supported for grouping purposes.

What you can use this for?

- analyze the coverage status by risk level (e.g. *How are my "Severe" level requirements/stories in this version? And in this environment? Are they all OK? Are some NOK due to failed tests?*)
- find important uncovered requirements (e.g. *Are all my requirements/stories with a high risk covered with test cases?*)
- know where you should spend your focus, i.e. prioritize your effort (e.g. "if all the more risky requirements have been addressed first")

Overall Requirement Coverage Report

The Overall Requirement Coverage Report allows visually grouping of requirements by custom fields, thus you may easily group by risk level related field, for example.

Calculator
 Calculator Board Enhanc...

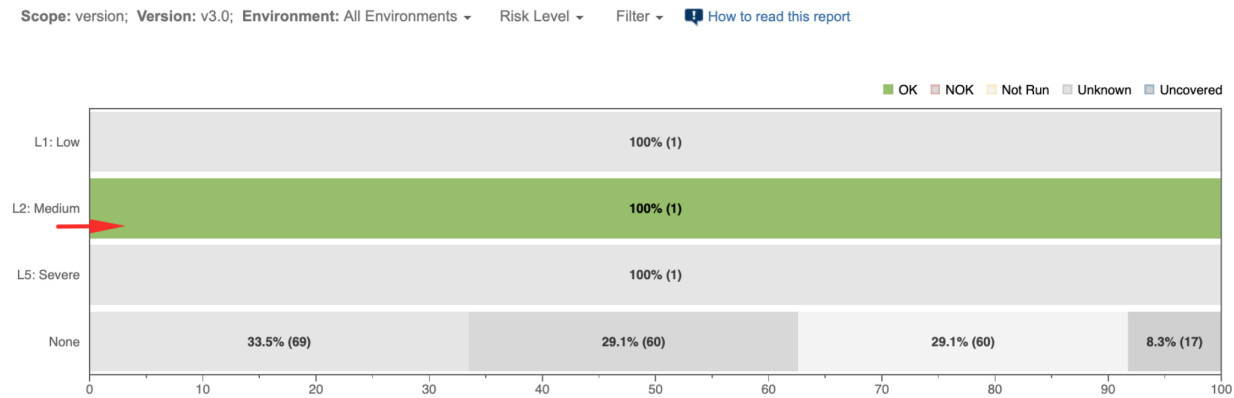
[Backlog](#)
[Active sprints](#)
[Releases](#)
[Reports](#)
[Issues](#)
[Components](#)
[Structure](#)
[Xray Reports](#)
[Risks](#)
[Xray Test Repository](#)
[Xray Test Plan Board](#)
[Automated Steps Library](#)

Overall Requirement Coverage Report [Switch report](#)



The report also allows you to drill-down on the bar and thus see exactly which requirements have that risk level, for example, and that are on that specific coverage status (e.g. "OK"). This can be quite handy to evaluate the completeness and the failed tests, especially if the requirement is NOK.

Overall Requirement Coverage Report [Switch report](#)



Requirements with Risk Level "L2: Medium" and Status "OK"

[View Issues](#)

Show 10 entries

Key	Summary	Total Tests	Tests Passed	Tests Failed	Tests Unknown	Completeness
CALC-3026	As a user, I can calculate the sum of two numbers	3	3	0	0	100%


Showing 1 to 1 of 1 entries


First Previous 1 Next Last

Overall Requirement Coverage gadget

If using the Overall Requirement Coverage gadget, then you need to pick a project (and not a saved filter) as the data source; only then Group By will allow you to select specific custom fields to group your requirements, such as the Risk Level related custom field.


Overall Requirement Coverage: (v3.0)

Project or Filter: 

Group by: 


Choose a field for grouped Requirements


☐ Flat Requirement Presentation

Execution Scope: 


Choose the Execution Scope to analyse the requirements

Test Plan Key:

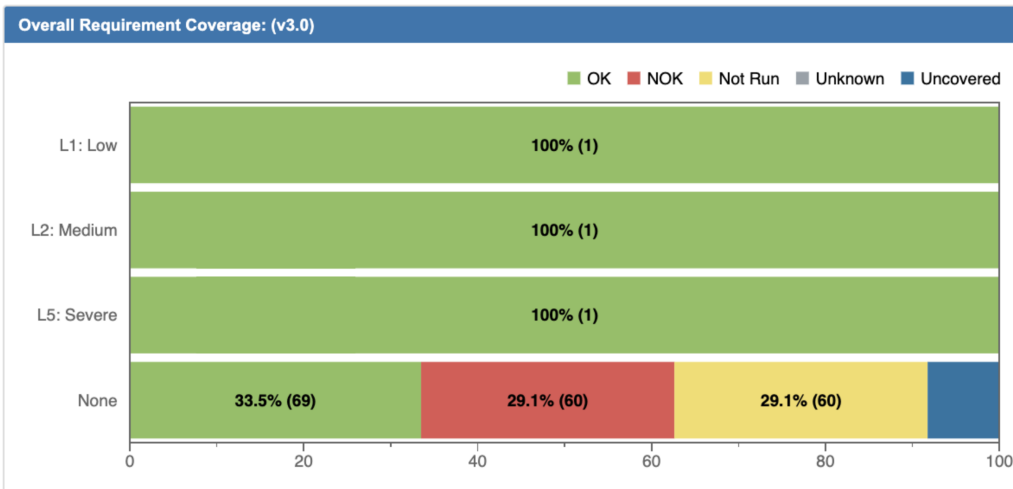
Project: 

Analysis Version: 

Choose the project Version to analyse

Refresh Interval: 

How often you would like this gadget to update



Requirements List gadget

The Requirements List gadget may be used to show just the coverage status, completeness and information about the passed/failed tests of requirements having a certain risk level. Thus, it is a great way to track the actual consolidated progress of your requirements/stories.

As this gadget does not provide a grouping mechanism, you need to instantiate it multiple times configuring each instance with a different filter for picking only the requirements having a certain risk level.

Requirements List: RR_risks_high - (No Version)(All Environments)

Custom Title:

☐ Use a custom title

Saved Filter: **RR_risks_high**

Project: **RR_risks_high**

Execution Scope: **Version**

Test Plan Key:

Analysis Version: **No Version**

Test Environment: **All Environments**

Requirement Status: **OK**

Choose the requirement status to analyze

Requirements List: RR_risks_high - (No Version)(All Environments)

Key	Summary	Status	Fix Version	Total	Passed	Failed	Others	Completeness
CALC-4665	replacing relational database with a NoSQL one for authentication	NOTRUN		2	1	0	1	50%
CALC-4664	replacing AngularJS	UNCOVERED		0	0	0	0	0%

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

Requirements List: RR_risks_medium - (No Version)(All Environments)

Key	Summary	Status	Fix Version	Total	Passed	Failed	Others	Completeness
CALC-4663	opensource library dependency	UNCOVERED		0	0	0	0	0%

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Analyzing testing results by risk level

Consolidated results

None of the following gadgets provide a grouping mechanism, thus you need to create multiple instances of them configuring each one with a different filter for picking only the Tests having a certain risk level.

Whenever analyzing the consolidated results, you may evaluate how your Tests are on a specific project version or just present their latest results (no version specified in the "Analysis Version" configuration field).



Please note

Coverage calculation in Xray takes into account the testing results. Therefore, it's possible to indirectly analyze it by looking at the requirements and their coverage status.

Overall Test Results gadget

Overall Test Results: tests_risk_high - scope(v3.0)

Custom Title:

☐ Use a custom title

Test Filter

No Filter selected

tests_risk_

tests_risk_high

tests_risk_low

tests_risk_medium

tests_risk_severe

tests_risk_very_high

Execution Scope

Project

Analysis Version

v3.0

Choose the project Version to analyse

Test Plan Key

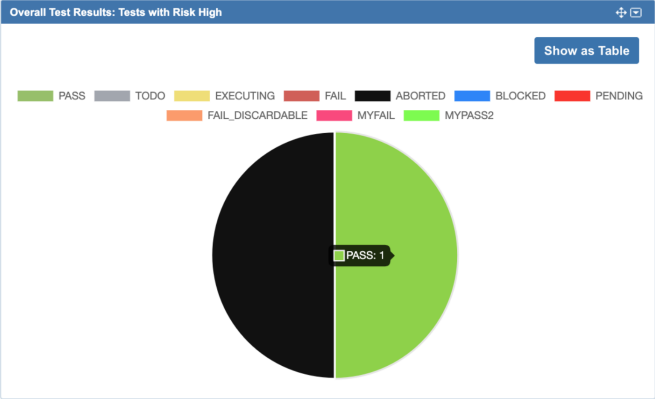
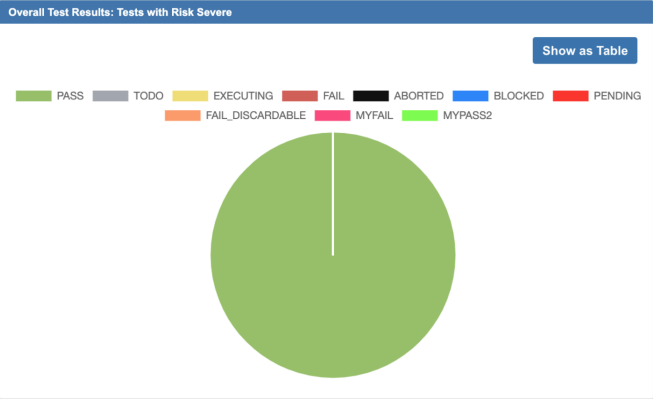
Refresh Interval:

Never

How often you would like this gadget to update

Save

Cancel



Tests List gadget

Tests list: Tests with Risk Severe

Custom Title:
☒ Use a custom title

Saved Filter: **No Filter selected**

Execution Scope:

Project:

Test Plan Key:

Analysis Version:
Choose the project Version to analyse

Test Environment:
Choose the Test Environment to analyse

Test Component:
Choose the project component to analyze

Test Priority:
Choose the Test Priority

Number of results:
Number of results to display

Refresh Interval:

Tests list: Tests with Risk Severe			
Key	Summary	Components	Status
CALC-3703	ability to perform clear / start new operations fr...		PASS
Showing 1 to 1 of 1 entries			
First Previous 1 Next Last			

Tests list: Tests with Risk High			
Key	Summary	Components	Status
CALC-3700	sample test		ABORTED
CALC-1202	CanAddNumbers		PASS
Showing 1 to 2 of 2 entries			
First Previous 1 Next Last			

Tests report (legacy report on project tab)

The legacy Tests report, available on the project tab, can be configured to show additional sortable columns. Thus, you may add the risk level related field (or others) and sort by it.

Key	Summary	Test Type	Risk Level	Status
CALC-3706	check the history of operations	Manual	L1: Low	v3.0 - TODO
CALC-1203	CanDoStuff	Generic	L2: Medium	v3.0 - PASS
CALC-1202	CanAddNumbers	Generic	L3: High	v3.0 - PASS
CALC-3700	sample test	Manual	L3: High	v3.0 - TODO
CALC-3703	ability to perform clear / start new operations from scratch	Manual	L5: Severe	v3.0 - TODO
CALC-4630	Does not do much!	Generic		v3.0 - FAIL
CALC-4625	My second test Gets, types and asserts	Generic		v3.0 - PASS
CALC-4626	My First Test Does not do much!	Generic		v3.0 - PASS
CALC-4617	Test story1 - t2	Manual		v3.0 - PASS
CALC-4616	Test story1 - t1	Manual		v3.0 - PASS

Note: this report is still a bit limited, so no advanced/flexible filtering mechanisms are available.

Finding out defects related with certain risk levels

As your testing progresses, you may find certain defects along the way that you have to manage. You should classify your defects or even deal with them as risks.

Anyway, you'll need to find defects related with certain assessed risks, no matter at what level they were identified, so you can decide how to proceed or treat them.

Xray provides several JQL functions that can be used to obtain these defects. Remember to complement your JQL query so you can find only the defects you want (e.g. the ones assigned to a certain AffectsVersion, for example).

Examples of JQL queries:

- risks defined at requirement or project level
 - issue in **defectsCreatedForRequirement**("risks_severe") and ...
- risks defined at Test level
 - issue in **defectsCreatedDuringTesting**("tests_risk_severe") and ...

Track and/or disallow changes on "requirements" and in test cases

In order to avoid unnecessary risks or at least mitigate them, you may enforce reviewing on requirements and even on test cases. Since they're issue type based, you may apply Jira workflows on them and/or set them read-only. This can be extended to Test Plans and any other issue types used by Xray.

Usage examples:

- upon a change on a "requirement"/story, enforce related Tests to be reviewed (i.e. transition the related Tests to a specific workflow status). Check [ScripRunner implementation example](#)
- set "requirement"/story related issue to read-only on a specific workflow transition, to ensure no changes are made without being reviewed; in order to make changes, require a specific workflow transition to be made; to implement this you just need to configure the related Jira workflow
- enforce workflows on Test and Pre-Condition issues, to make sure the specification is reviewed. More information about using workflows on Xray entities, [here](#)

Note: all changes are tracked, directly on the respective issues; in the case of Test Runs, changes also get tracked on the related Activity section.

Use different Test Plans based on Risk level

Xray gives you the ability to create one or more Test Plans if you wish to manage and track the testing progress for different tests independently.

Therefore you may create, for example, one Test Plan containing all the Tests related with the highest risk level items and distinct ones for the other risk levels; on each Test Plan you may further prioritize Tests.

If you have multiple Test Plans, and since they're issues, you may prioritize them in the current Sprint or Kanban boards. You can also "label" these Test Plans somehow to distinguish their relative relevance; you may use Jira's native Priority field for this as it comes by default and fulfills this purpose.

Setup and Configuration

In order to start with RBT, you have to [configure the risk management related fields](#). You may use Jira built-in capabilities or use a risk management or even a customization app instead.

Either way, later on, we would recommend using saved filters as a way to quickly obtain Tests or the issues where the risk is being defined on.

Configuring saved filters

Risk at Test level

If you're adopting risk at Test level, you may create several saved filters (e.g. "tests_risk_low", "tests_risk_medium", "tests_risk_high", "tests_risk_very_high", "tests_risk_severe") as this can later be quite useful, namely for using within gadgets or reports.

tests_risk_high

Save as

Details

★

Share

Export

Tools

project = CALC AND issuetype = Test AND "Risk Level" = "L3: High"

?

Q

Basic

Columns

1-2 of 2

T	Key	Summary	P	Status	Created	Updated	Risk Level	Risk	Impact	Probability	
<div>🔍</div>	CALC-3700	sample test	<div>🚫</div>	OPEN	06/Feb/19	03/May/19	L3: High	9	3	3	...
<div>🔍</div>	CALC-1202	CanAddNumbers	<div>🚫</div>	OPEN	18/Apr/17	03/May/19	L3: High	8	4	2	

1-2 of 2

Risk at requirement or at project level

If you're using adopting risk at requirement/story or at project level instead, you may follow a similar approach to group those issues by risk level.

Create a set of filters, each one related with a different risk level. Using a nomenclature for the saved filters can help you out. Example: "RR_risks_low", "RR_risks_medium", "RR_risks_high", "RR_risks_extreme".

RR_risks_high

Save as

Details

★

project = CALC AND issuetype = Risk AND exposure = High

?

Q

Basic

Columns

1-2 of 2

T	Key	Summary	P	Status	Created	Updated	RR: Impact	RR: Probability	Exposure	
	CALC-4665	replacing relational database with a NoSQL one for authentication	🔴	OPEN	03/May/19	03/May/19	Major	Almost certain	HIGH	...
	CALC-4664	replacing AngularJS	🔴	OPEN	03/May/19	03/May/19	Catastrophic	Very likely	HIGH	

1-2 of 2

To make our life easier, we may also use saved filters to group Tests by the indirectly associated risk level, using JQL and the **requirementTests()** function.

tests_from_requirements_risk_high

Save as

Details

★

issue in requirementTests(RR_risks_high)

?

Q

Basic

Columns

1-2 of 2

T	Key	Summary	P	Status	Created	Updated	
	CALC-4667	authenticate with an unauthorized user (e.g. invalid, blocked)		OPEN	03/May/19	03/May/19	...
	CALC-4666	authenticate with a valid user		OPEN	03/May/19	03/May/19	

1-2 of 2

You won't need these pre-created filters to pick tests sorted by risk level whenever adding Tests to a Test Execution/Test Plan as you may write JQL query inline; however, they will be required if you aim to filter out Tests in gadgets and in reports by the related risk level.

Configuring Risk, Probability and Impact

The first thing you have to do is to configure risk (level), probability and impact. Xray does not provide any of these fields; thus, you have to either configure them by yourself or use an app to assist you with that.

The only thing that you'll need in the end is the risk level related one because that's the one you will use to pick tests sorted by risk and to execute them by that order.

Risk, probability and impact may be configured as "Number," "Text Field (single line)" or "Select List (single choice)" based custom fields.

Always have in mind how sorting works for that field as this is critical for RBT.



Please note

"Number" based custom fields are naturally sortable and thus are the easiest ones to implement; however, they're not very user-friendly.

If you use "Text Field (single line)", note that the sorting will be based on text/string comparison (thus, for example, the text "Medium" will be greater than "High.") In this case, you may add a prefix in order to guarantee the correct order (e.g. "L2: Medium", "L3: High"). Using "text fields", namely for the risk level, may be tricky as it will make it harder to search for them; thus, we don't recommend their usage.

On the other hand, "Select List (single choice)" based fields have options (i.e. well-defined possible values) along with a respective order; therefore, this order will be the one used whenever ordering issues by this type of fields. "Select List (single choice)" fields are also easier to use during filtering, as you don't have to type their values by hand, besides giving you the ability to filter by multiple values at the same time. However, if you decide to configure your risk level as a "Select List (single choice)" field, you'll need to do some additional work to calculate it based on the probability and impact fields.

Note that an empty value (i.e. None) for these fields will have a higher ranking than all other options; thus, you may set an additional option (e.g. N/A) as the first/lowest ranked one and set it as the default value.



Disclaimer

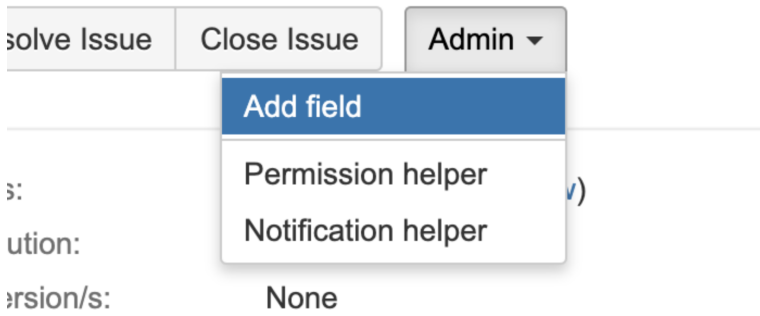
All configuration examples, including related source-code, provided herein using apps from other vendors are just informative and need to be evaluated properly, considering also performance impacts among other. We don't provide support for these apps neither for their configuration.

Using Jira natively

Jira can be used to define risk, probability and impact; however, notice that Jira does not provide an out-of-the-box solution for calculating the value of a field (i.e. risk level) based on the values of other fields. You'll need to either make a custom development for that or use an app to help you out.

Whenever creating fields, please have a look at the previous note on their types. We advise you to set this first in a testing environment to make sure it works as expected and covers the needs of the teams adopting RBT.

After choosing the type of fields you'll need, adding/creating them is straightforward; this can be done right from the issue screen using **Admin > Add field**.

A screenshot of the 'Add Field' dialog box. It has a title bar 'Add Field'. Inside, there is a search bar with the text 'Impact' and a dropdown arrow. Below the search bar, it says 'Search for an existing field, or name a new one.' At the bottom right, there are two buttons: 'Add field' and 'Cancel'.

It can also be done from Jira's administration, under **Issues > Custom fields > Add custom field**.

A screenshot of the Jira Administration page. The left sidebar shows the 'Administration' header and a search bar. Below it are tabs for 'Applications', 'Projects', 'Issues', 'Add-ons', 'User management', 'System', and 'Structure'. The 'Issues' tab is selected. Under 'Issues', there are links for 'Issue types', 'Issue type schemes', 'Sub-tasks', 'WORKFLOWS', 'Workflows', 'Workflow schemes', 'SCREENS', 'Screens', 'Screen schemes', 'Issue type screen schemes', and 'FIELDS'. The 'Custom fields' link under 'FIELDS' is highlighted with a red box. The main content area shows the 'Custom fields' page. It has a table with columns: 'Name', 'Type', 'Available Context(s)', and 'Screens'. The table lists three custom fields: 'Asset' (Text Field (single line)), 'AttributeX' (Text Field (single line)), and 'Begin Date' (Date Time Picker). The 'Add custom field' button is highlighted with a red box.

Configuration examples

Impact/Probability

You may create a custom field "Select List (single choice)" based to store and manage impact and probability related fields. You may also use "Number" based fields instead.

If you use a "Select List" then you need to make sure their option values are sorted in the right order and that their values are also alphabetically sorted in the same way.

Configure Custom Field: Impact

Below are the Custom Field Configuration schemes for this custom field. Schemes can be configured differently for each project context or in a global context. Moreover, project level configuration is available.

- [Add new context](#)
- [View Custom Fields](#)

Default Configuration Scheme for Impact

Default configuration scheme generated by JIRA

Applicable contexts for scheme: [Edit Configuration](#)

Issue type(s):
Global (all issues)

Default Value: [Edit Default Value](#)

Options: [Edit Options](#)

- 0
- 1
- 2
- 3
- 4

In the previous example, 0 is redundant with having an empty value; thus, you may either set the default to be 0 (instead of "none") or remove the "0" option.

Risk level

You may create a custom field, either "Number" or "Select List (single choice)" based, to store and manage your risk. A "Select List" may provide additional benefits.

Select a Field Type

This custom field will also be visible in 3 other projects.

All

Standard

Advanced

1

2

Select List (cascading)

Choose multiple values using two select lists.

Option 1

Option 2

Select List (multiple choices)

Choose multiple values in a select list.

Select

Select List (single choice)

A single select list with a configurable list of options.

Find more custom fields

Previous

Next

Cancel

Configure 'Select List (single choice)' Field

Name *

Risk Level

Description

Options *

Add

L1: Low

x

L2: Medium

x

L3: High

x

L4: Very High

x

L5: Severe

x

Previous

Create

Cancel

No matter your choice, the risk level field won't be automatically calculated based on the probability/impact fields; you'll need to use a app or custom development for that as shown in the next sections.

Using Risk Management for Jira app

[Risk Management for Jira app](#) provides a simple, yet flexible approach for risk management, allowing you to assess risks in your existing issues.

Integration with Xray is straightforward as this app uses custom fields for the probability and the impact (i.e "consequence"). The risk level (i.e. "risk number") is also a custom field calculated automatically based on the previous ones.

Configuring fields

To configure Risk Management for Jira, go to **Add-ons > Risk Management > Required > Custom Fields**.

Add-ons
User management
System
Structure

RISK MANAGEMENT

About
Add-on info
Configuration
Template Guide
Risk Index Value Guide

REQUIRED

Custom Fields

EXTENDED
Column Order
Risk Colors
Risk Matrix
Searches
Tabs

ADVANCED
Labels
Style
Templates
Risk Index Value

Custom Fields

You must create two Custom Fields for the Propability and Consequence values.

1 You can have the Add On create, add and configure the fields at [Add Custom Fields](#)

It is strongly recommended to use custom fields of type *Select List*. Number fields and text fields will probably work, but the behavior is not documented and not supported.

Probability Field*

Manual Test Steps (Export)
NNNNN
Pre-Condition Type
Pre-Condition association wit
Probability

VALID

The Custom Field that holds the probability (Likelihood) value

Consequence Field*

Epic Name
Epic Status
Epic/Theme
Flagged
Generic Test Definition
Impact

VALID

The Custom Field that holds the Consequence (Impact) value

Risk Number*

ADDED RISKINDEXNUMBERFIELD

The Add On specific Custom Field (RiskManagementCalculatedNumberField or RiskIndexNumberField) that holds the calculated value of the Probability and Consequence field.

Save
Cancel

In the previous screen you can choose to reuse existing fields for abstracting the probability and the impact (they should be "Select List" based).

It can also create these fields for you as well, along with the risk level. Choose "Add Custom Fields" and the name for the impact (i.e. consequence), probability and risk level (i.e. risk index) fields; please check "RiskIndexNumberField" flag.

Add Custom Fields

You can have the Add On add the required custom fields to Jira and the Add On configuration. Select a Field Screen Tab and specify names for the fields.

Leave the name input empty if that particular field should not be added.

Field Screen*

AI: Software Development Bug Scree
Field Tab
AI: Software Development Default Iss
Field Tab
AI: Software Development Resolve Is
Field Tab
AP: Software Development Bug Scree
Field Tab

Consequence Field

Risk Impact

VALID

State a name for the Custom Field that holds the Consequence (Impact) value

Probability Field

Risk Likelihood

VALID

State a name for the Custom Field that holds the probability (Likelihood) value

Risk Index Field

☐ Use RiskManagementCalculatedNumberField
☒ Use RiskIndexNumberField (Recommended)

Risk Score

ADDED RISKINDEXNUMBERFIELD

State a name for the Custom Field that holds the Risk Index value

Save
Cancel




Please note

If you already had values for probability and impact/consequence, indexing will not recalculate the risk level, as the app uses a listener to update it upon changes.

Using it

On issues having the impact/consequence and probability related fields, risk level ("Risk Score" on the following example) will be updated upon update on any of these fields.

 Calculator / CALC-3706

check the history of operations

Edit

Comment

Assign

More

Start Progress

Resolve Issue

Close Issue

Admin

Details

Type: Test

Priority: Blocker

Affects Version/s: None

Component/s: None

Labels: lixo

Impact: 2

Probability: 1

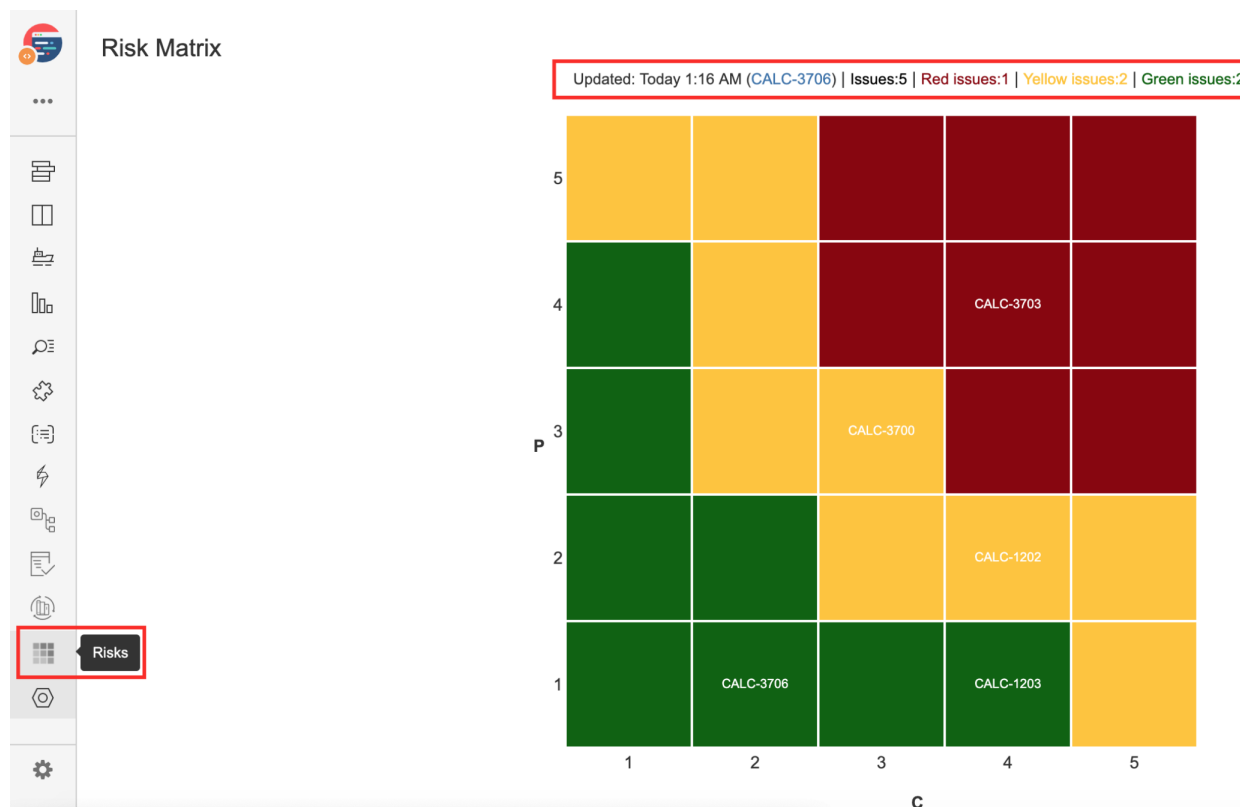
Status: OPEN (View Workflow)

Resolution: Unresolved

Fix Version/s: None

Risk Score: (2)

Within the Risk Matrix, issues can easily be found/searched as they're mapped to a cell and to a specific color, depending on their probability, impact /consequence and calculated risk level.



In Xray, whenever adding Tests to a Test Execution for example, the risk level field (e.g "Risk Score") can later be used to sort Tests ascending /descending; it can also be used as a filter.

Add Tests to Test Execution CALC-3702

AssigneeIssue Assignee

SelectSearchJQL

Filter(s)

Project

Calculator (CALC) x

Test Type

Choose the Test Type

Contains text

Risk Score

between

2

and

16

The Risk Score Field for Risk Management

Clear

Search

Issue Type

Key

Summary

Risk Level

Probability

Impact

Risk Score

CALC-3703

ability to perform clear / start new operations from scratch

L5: Severe

4

4

(((16)))

CALC-1202

CanAddNumbers

L3: High

2

4

((8))

CALC-1203

CanDoStuff

L2: Medium

1

4

(4)

CALC-3706

check the history of operations

L1: Low

1

2

(2)

Showing 1 to 4 of 4 entries

First

Previous

1

Next

Last

☒ Hide Tests in non-executable Statuses

Add selectedAdd all (4)Cancel

Using Risk Register app

Risk Register app provides risk management at project level, by allowing you to create and manage Risks as issues.

i

Please note

Although it may be possible to assess risks in your existing issues by adding the fields to them, this seems to be an unsupported featured by Risk Register; therefore we don't recommend using it, unless stated otherwise by the app vendor.

Risk Register app uses a set of custom fields (e.g. Impact, Probability, Exposure, etc) that will be created during its setup.

Configuring it

1. configure Risk Register (fields, screens, issue type)

a. configure global settings of the app (Add-ons > Risk Register > Settings), including the custom fields used by Risk Register, which are always created (you may change their name afterwards). If you have existing custom fields with similar names, new ones will be created (no existing fields will or can be reused).
You need also to define which issue type will be used to abstract the risk; Risk Register can create one for you (e.g. "Risk").

Administration Search JIRA admin

Applications Projects Issues **Add-ons** User management System Structure

ATLASSIAN MARKETPLACE
Find new apps
Manage apps

APWIDE
Configure Projects
Application Schemes
Environment Categories
Environment Status
Applications
Environments
Environment Properties
Attributes
Teams
Roles
Permission Schemes

RISK REGISTER
Getting started
Risk registers
Exposures
Risk model
Settings
Attribution

Add-ons / Risk Register
Getting started
Risk Register is an app that adds risk management features to Jira.
The set-up for Risk Register involves two tasks:

- Adjust settings**
Review the **add-on settings** and address any errors reported there. [Read More...](#)
- Add a risk register**
Add a risk register to an existing Jira project. ... [Read More](#)

After the above set-up tasks have been completed, you can create risks as you would any other type of Jira issue.

Create Issue

Project* Aye Bee Cee (ABC)
Issue Type* ☒ Risk ☐ Task
Next Cancel

Applications Projects Issues **Add-ons** User management System Structure

ATLASSIAN MARKETPLACE
Find new apps
Manage apps

APWIDE
Configure Projects
Application Schemes
Environment Categories
Environment Status
Applications
Environments
Environment Properties
Attributes
Teams
Roles
Permission Schemes

RISK REGISTER
Getting started
Risk registers
Exposures
Risk model
Settings

Risk Register - settings

FIELDS ISSUE TYPE SCREENS ⚠️ CONTROLS TREATMENT WORKFLOW

- ✓ RR: Impact
- ✓ Residual Impact
- ✓ RR: Probability
- ✓ Residual Probability
- ✓ Exposure
- ✓ Residual Exposure
- ✓ Treatment

- b. enable risk management for your project, either from the project settings "Risk register" section or from **Jira administration > Add-ons > Risk Register > Risk registers**.

Project settings

Summary
Details
Re-index project
Delete project

Issue types
Bug
Capability
Epic
Improvement
Initiative
New Feature
Pre-Condition
Requirement
ResultSet
Risk
13 more issue types

Workflows
Screens
Fields

Versions
Components
Risk register

Risk Register Add-on settings

Enable risk management for project CALC

- ✓ **Issue type**
The 'Risk' issue type is available in this project.
- ✓ **Screens**
This project uses the screen scheme 'Default Xray Test Screen Scheme' for 'Risk' issues.
- ✓ **Fields**
All essential fields are visible in the 'Default Field Configuration' field configuration.
- ✓ **Workflow**
This project uses the workflow 'classic default workflow' for 'Risk' issues.

Administration Search JIRA admin Back to project: Calculator

Applications Projects Issues **Add-ons** User management System Structure

ATLASSIAN MARKETPLACE
Find new apps
Manage apps

APWIDE
Configure Projects
Application Schemes
Environment Categories
Environment Status
Applications
Environments
Environment Properties
Attributes
Teams
Roles
Permission Schemes

RISK REGISTER
Getting started
Risk registers

Risk registers Calculator [+ Add a register](#)

2. in Xray, configure Risk as a "requirement issue type" (i.e. a coverable issue type that you can cover with test cases)
 - a. Some of the risks that you have identified during risk assessment may be covered with tests. Therefore, they can be handled similar to "requirements" in Xray. Just go to your **Jira administration > Add-ons > Xray > Issue Type Mapping** and drag the Risk issue type to the "Requirement Issue Types" column and save it. The project containing Risk issues must be enabled as a requirements project, either from **Jira administration > Add-ons > Xray > Requirement Projects** or from the project settings page, within the Summary section, using the action "Enable Xray Requirement Coverage."

The screenshot shows the Jira Administration interface for the Xray add-on, specifically the 'Issue Type Mapping' section. On the left, there's a sidebar with navigation links like 'Applications', 'Projects', 'Issues', 'Add-ons', 'User management', 'System', and 'Structure'. The main content area is titled 'Issue Type Mapping' and includes a warning message: 'We recommend that you perform a re-index operation due to configuration changes.' Below this, there are three columns: 'Available Issue Types', 'Requirement Issue Types', and 'Defect Issue Types'. In the 'Available Issue Types' column, 'Risk' is highlighted with a red box and a red arrow pointing to the 'Requirement Issue Types' column, where it has been moved. The 'Defect Issue Types' column contains 'Bug'.

Using it

Some of the identified project level risks may be mitigated with tests while other ones may not.

For those that can, you may start by defining the "Risk assessment" related fields, including probability and impact, in the Risk issues (i.e. the ones whose issue type is configured to be handled as risk).

The screenshot displays the Xray interface for a specific issue, CALC-4665, which is of type 'Risk'. The top section shows the issue details, including its status (OPEN), priority (Major), and resolution (Unresolved). Below this, the 'Description' section is visible. The 'Test Coverage' section shows a table with two test cases, both of which are 'OPEN' and 'Unresolved'. The 'Risk assessment' section is also visible, showing a 'Specify Residual Exposure' toggle and a 'Risk register' button. The 'Exposure' section shows a horizontal bar chart with a red bar indicating a 'HIGH' risk level. The chart is labeled 'RR: Probability: Almost certain' and 'RR: Impact: Major'.

Later on, we'll need to find these Tests based on the "exposure" of the associated/covered risk. To make our life easier, we may use saved filters to group Tests by the indirectly associated risk exposure.

Create a set of filters, each one related with a different "Exposure" (i.e risk level). Using a nomenclature for the saved filters can help you out. Example: "RR_risks_low", "RR_risks_medium", "RR_risks_high", "RR_risks_extreme".

RR_risks_high Save as Details ★ Share Export Tools

project = CALC AND issuetype = Risk AND exposure = High

1-2 of 2

T	Key	Summary	P	Status	Created	Updated	RR: Impact	RR: Probability	Exposure	Columns
🔍	CALC-4665	replacing relational database with a NoSQL one for authentication	🔴	OPEN	03/May/19	03/May/19	Major	Almost certain	HIGH	...
🔍	CALC-4664	replacing AngularJS	🔴	OPEN	03/May/19	03/May/19	Catastrophic	Very likely	HIGH	

1-2 of 2

To obtain the related Tests, you can use JQL and the **requirementTests()** function.

Thus, lets say you want to obtain all Tests related with Risks with an "High" exposure in order to add them to an existing Test Execution; you can do as follows.

Add Tests to Test Execution CALC-3702 Assignee Issue Assignee

Select Search JQL

JQL Search
issue in requirementTests("RR_risks_high")

Search

Columns

Issue Type	Key	Summary	Priority
🔍	CALC-4666	authenticate with a valid user	🔴
🔍	CALC-4667	authenticate with an unauthorized user (e.g. invalid, blocked)	🔴

Showing 1 to 2 of 2 entries First Previous 1 Next Last

The returned Tests may have different relevance; therefore, if you want, you may use a complementary field (e.g. "Priority") to sort them out.

If you wish, you may also assess Risk at Test level and define probability, impact and risk level on the Test issues. Then you may sort them by their individual risk level.

Using Jira Misc Custom Fields app

[Jira Misc Custom Fields](#) (JMCF) is a popular app for implementing calculated custom fields very easily. JMCF provides a set of types that can be used, for example, to implement Select List based fields (i.e. "dropdown").

If you already have JMCF, you may use it to perform the calculation of risk level.

Impact and probability may be created as standard Jira custom fields, using for example the standard "Number" or even "Select List (single choice)" based fields; for those you won't need this app.

For the calculation of risk you may use a Calculated Single-select" field (recommended) or a "Calculated Number" field from JMCF.

Select a Field Type

This custom field will also be visible in 3 other projects.

All
Standard
Advanced

Calculated (scripted) Number Field (JMCF app)
A custom field type that displays a Number calculated using a Groovy script. The custom field must be configured from the Custom Fields page after creation.

Calculated (scripted) Single-select Field (JMCF app)
A custom field type that displays a single Option value calculated using a Groovy script. Unlike a calculated text field, it allows for custom ordering of values and guided search, and it can be used in statistics dashboard gadgets. The custom field must be configured from the Custom Fields page after creation.

Calculated (scripted) Single-User Field (JMCF app)
A custom field type that displays a User calculated using a Groovy script. The custom field must be configured from the Custom Fields page after creation.

[Find more custom fields](#)
[Previous](#)
[Next](#)
[Cancel](#)

After creating the calculated field, you'll need to perform re-indexing if your project already has values for probability and impact.

Calculating risk at Test level and storing it in a Calculated Single-select field

In this approach, we're using a Calculated Single-select field to present the risk level.

The first thing we need to do is filling out the possible options (e.g. "L1: Low", "L2: Medium", etc). Then we can specify the groovy script that is used to calculate the value, that must be one of the predefined options.

Using a single-select field will ease the search of issues later on.

Calculated (scripted) Single-select Field
Powered by

A custom field type that displays a single Option value calculated using a Groovy script. Unlike a calculated text field, it allows for custom ordering of values and guided search, and it can be used in statistics dashboard gadgets.

[Help with this feature](#)
[Explore Documentation](#)
[Atlassian Community](#)
[Get support](#)

Configure Custom Field: JMCF - Risk Level

Below are the Custom Field Configuration schemes for this custom field. Schemes are applicable for various issues types in a particular context. You can configure a custom field differently for each project context or in a global context. Moreover, project level schemes will over-ride global ones.

- [Add new context](#)
- [View Custom Fields](#)

Default Configuration Scheme for JMCF - Risk Level

Default configuration scheme generated by JIRA

Applicable contexts for scheme: [Edit Configuration](#)

Issue type(s):
Global (all issues)

Groovy Formula: [Edit Groovy Formula](#)

The value of this field will be calculated using the following formula (Groovy script):

```

impact = issue.get("Impact")
probability = issue.get("Probability")

def risk
def level
if ((impact) && (probability)) {
    risk = Double.parseDouble(impact) * Double.

```

Options: [Edit Options](#)

- L1: Low
- L2: Medium
- L3: High
- L4: Very High
- L5: Severe

Search Template: [Edit Search Template](#)

Multi Select Searcher
 This is a global option for the field. Any change will be applied to all configuration schemes of this field.

Velocity Template: [Edit Velocity Template](#)

This field will be displayed using the default Velocity template.

Show on Edit Screen: ☐ Never show on Transition and Edit screens
If checked, the field will not be visible on Edit and Transition screens even if added to them.

Groovy Formula for Risk (Level) using a Calculated Single-select Field

```
impact = issue.get("Impact")
probability = issue.get("Probability")

def risk
def level
if ((impact) && (probability)) {
    risk = Double.parseDouble(impact) * Double.parseDouble(probability)
} else {
    risk = null
}

if ((risk>0) && (risk<4)) {
    level = "L1: Low"
} else if ((risk>=4) && (risk<8)) {
    level = "L2: Medium"
} else if ((risk>=8) && (risk<12)) {
    level = "L3: High"
} else if ((risk>=12) && (risk<16)) {
    level = "L4: Very High"
} else if (risk==16) {
    level = "L5: Severe"
} else {
    level = null
}

return level
```

Using it

After creating the previous fields, you may use it, for example, in the dialog for adding Tests to an existing Test Execution; you can also easily filter issues by risk level.

The screenshot shows the 'Add Tests to Test Execution' dialog for test execution CALC-3702. The dialog includes a table of issues with columns: Issue Type, Key, Summary, Probability, Impact, and JMCF - Risk Level. The 'JMCF - Risk Level' column is highlighted with a red box, and a dropdown menu is open showing the risk levels: L1: Low, L2: Medium, L3: High, L4: Very High, and L5: Severe. A red arrow points to the 'JMCF - Risk Level' column header.

Issue Type	Key	Summary	Probability	Impact	JMCF - Risk Level
BUG	CALC-3703	ability to perform clear / start new operations from scratch	4	4	L5: Severe
BUG	CALC-1202	CanAddNumbers	2	4	L3: High
BUG	CALC-1203	CanDoStuff	1	4	L2: Medium
BUG	CALC-3706	check the history of operations	1	2	L1: Low

Calculating risk and storing it in a Calculated Number field

In this approach, risk is calculated and stored in a Calculated Number field; we just need to define the groovy script that will be used to calculate it.

Calculated (scripted) Number Field

A custom field type that displays a Number calculated using a Groovy script.

Powered by  **JMCF**

[Help with this feature](#) [Explore Documentation](#) [Atlassian Community](#) [Get support](#)

Configure Custom Field: JMCF - Risk Value



Below are the Custom Field Configuration schemes for this custom field. Schemes are applicable for various issues types in a particular context. You can configure a custom field differently for each project context or in a global context. Moreover, project level schemes will over-ride global ones.

- [Add new context](#)
- [View Custom Fields](#)



Default Configuration Scheme for JMCF - Risk Value

Default configuration scheme generated by JIRA

Applicable contexts for scheme: [Edit Configuration](#)

Issue type(s):
Global (all issues)

Groovy Formula: [Edit Groovy Formula](#)

The value of this field will be calculated using the following formula (Groovy script):


```
Integer.parseInt(issue.get("Impact") ? : 0) * Integer.parseInt(issue.get("Probability") ? : 0)
```

Format Expression: [Edit Format Expression](#)

The value of this field will be formatted according to Jira's default number format.

Search Template: [Edit Search Template](#)

Number range searcher (statistics-compatible)

 This is a global option for the field. Any change will be applied to *all* configuration schemes of this field.

Velocity Template: [Edit Velocity Template](#)

This field will be displayed using the default Velocity template.

Show on Edit Screen: ☐ Never show on Transition and Edit screens

If checked, the field will not be visible on Edit and Transition screens even if added to them.

Groovy Formula for Risk (Level) using a Calculated Number Field

```
impact = issue.get("Impact")
probability = issue.get("Probability")

def risk
if ((impact) && (probability)) {
    risk = Double.parseDouble(impact) * Double.parseDouble(probability)
} else {
    risk = null
}

return risk
```

Calculating risk at requirement/project level and storing it in a Calculated Single-select field at Test level

Although it could seem feasible to implement a calculated single-select field that would calculate the risk level based on the impact and probability defined on the requirement/project level risk, it would not be enough.

Why? First of all you need to make sure the calculation happens upon changes on the parent impact and probability values; second, you need to make sure the issue and the calculated value are indexed after it.

As JMCF does not provide a way to implement event listeners, there is currently no viable solution for this. As a possible alternative, please see a possible implementation using [ScriptRunner](#).

Using ScriptRunner

If you already have the ScriptRunner app, you may use it to perform the calculation of risk level.

Impact and probability may be created as standard Jira custom fields though, using for example "Number" or even "Select List (single choice)" based fields; for those you won't need ScriptRunner.

You may adopt different implementation approaches for calculating and storing the risk level; the following sections depict different scenarios. We recommend using a "Select List (single choice)" based field to persist the risk, as shown in the first scenario, as it will be easier to find/filter later on.

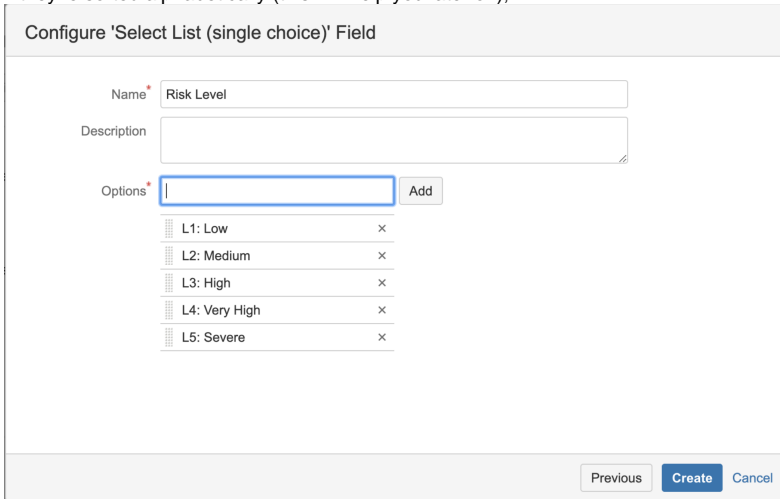
Calculating risk at Test level and storing it in a Single Select field using a Script Listener

In this example, we're calculating the risk level based on the changes made on issues (i.e. by listening on the "issue created" and "issue updated" Jira events). You can use a Script Field for this.

For optimization purposes, only issues of a certain issue type on a certain project will be thoroughly evaluated; from those, the script will only proceed if changes were made to the probability and impact related custom fields (their name is defined within the script itself). Upon success, the script will update an existing custom field where the risk level will be stored in.

What you'll need to do:

1. create a standard custom field (e.g. "Risk Level") of type "Select List (single choice)," having the options corresponding to the different risk level bands correctly ordered (e.g. "L1: Low", "L2: Medium", "L3: High", "L4: Very High", "L5: Severe"). Make sure the option values maintain their order if they're sorted alphabetically (this will help you later on);



Configure 'Select List (single choice)' Field

Name * Risk Level

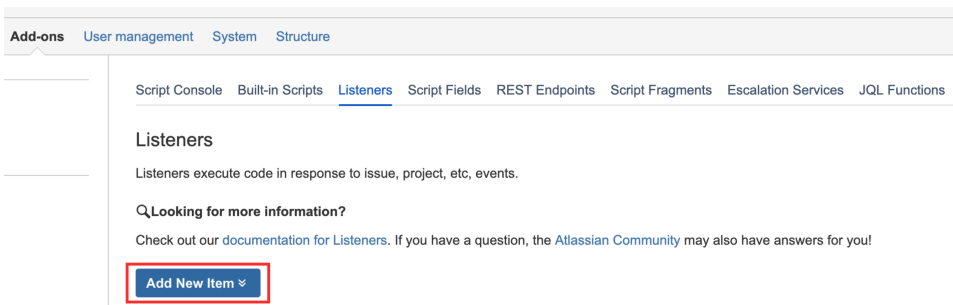
Description

Options * Add

- L1: Low x
- L2: Medium x
- L3: High x
- L4: Very High x
- L5: Severe x

Previous Create Cancel

2. using ScriptRunner, go to you Jira administration and create accustom Script Field (**Add-ons > ScriptRunner > Script Fields > Add New Item > Custom Script Field**).



Custom listener

Write your own groovy class as a custom listener.

Note

An optional note, used only for your reference.

Project(s)

Filter on events for these projects. Some events, eg **User** events, are not associated with a project.

Events

Which events to fire on .

Inline script

```
1 import com.atlassian.jira.issue.fields.CustomField
2 import com.atlassian.jira.issue.CustomFieldManager
3 import com.atlassian.jira.issue.MutableIssue
4 import com.atlassian.jira.issue.Issue
5 import com.atlassian.jira.component.ComponentAccess
6 import com.atlassian.jira.event.issue.IssueEvent
7 import com.atlassian.jira.issue.util.DefaultIssueC
8
```

Enter your script here

Script file

Path to the script accessible on the server - or fully-qualified class name class in the form com.acme.MyListener

Risk (Level) calculated using a Script Listener

```
import com.atlassian.jira.issue.fields.CustomField
import com.atlassian.jira.issue.CustomFieldManager
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.event.issue.IssueEvent
import com.atlassian.jira.issue.util.DefaultIssueChangeHolder
import com.atlassian.jira.issue.ModifiedValue
import com.atlassian.jira.issue.history.ChangeItemBean
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.issue.customfields.option.Option
import com.atlassian.jira.event.type.EventDispatchOption
import com.atlassian.jira.issue.index.IssueIndexingService
import com.atlassian.jira.util.ImportUtils

import org.apache.log4j.Logger
import org.apache.log4j.Level

def log = Logger.getLogger("com.onresolve.jira.groovy")
log.setLevel(Level.DEBUG)

MutableIssue issue = event.issue
def issueType = issue.issueTypeObject.name

// Only process Test or Story based issues (ADAPT IT tou your needs)
if (issueType != "Test" && issueType != "Story"){
    return
}

String cfImpactName = "Impact"
String cfProbabilityName = "Probability"
String cfRiskLevelName = "Risk Level"
```

```

def changeManager = ComponentAccessor.getChangeHistoryManager();
def changeLog = event.getChangeLog();

def impactChanged = false, probabilityChanged = false;

if ( changeLog != null ) {
    def changeId = changeLog.getLong( "id" );
    if ( changeId != null ) {
        def change = changeManager.getChangeHistoryById( changeId );
        for (ChangeItemBean bean : change.getChangeItemBeans() ) {
            if ( bean.getField().equals(cfImpactName)
                && ChangeItemBean.CUSTOM_FIELD.equals( bean.getFieldType() ) ) {
                log.debug(cfImpactName + " changed");
                impactChanged = true;
            }
            if ( bean.getField().equals(cfProbabilityName)
                && ChangeItemBean.CUSTOM_FIELD.equals( bean.getFieldType() ) ) {
                log.debug(cfProbabilityName + " changed");
                probabilityChanged = true;
            }
        }
    }
}

if (impactChanged || probabilityChanged){
    def issueManager = ComponentAccessor.issueManager
    def customFieldManager = ComponentAccessor.customFieldManager
    def cfImpact = customFieldManager.getCustomFieldObjectByName(cfImpactName)
    def cfProbability = customFieldManager.getCustomFieldObjectByName(cfProbabilityName)

    def option
    def impact
    def probability

    option = (Option)issue.getCustomFieldValue(cfImpact)
    if (option != null) impact = option.getValue()
    option = (Option)issue.getCustomFieldValue(cfProbability)
    if (option != null) probability = option.getValue()

    def risk
    def level

    if ((impact) && (probability)) {
        risk = Double.parseDouble(impact) * Double.parseDouble(probability)
    } else {
        risk = null
    }

    if ((risk>0) && (risk<4)) {
        level = "L1: Low"
    } else if ((risk>=4) && (risk<8)) {
        level = "L2: Medium"
    } else if ((risk>=8) && (risk<12)) {
        level = "L3: High"
    } else if ((risk>=12) && (risk<16)) {
        level = "L4: Very High"
    } else if (risk==16) {
        level = "L5: Severe"
    } else {
        level = null
    }

    def cfRiskLevel = customFieldManager.getCustomFieldObjectByName(cfRiskLevelName)
    def fieldConfig = cfRiskLevel.getRelevantConfig(issue)
    def value = ComponentAccessor.optionsManager.getOptions(fieldConfig)?.find { it.toString() == level }

    issue.setCustomFieldValue(cfRiskLevel, value)
    def currentUser = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
    issueManager.updateIssue(currentUser, issue, EventDispatchOption.DO_NOT_DISPATCH, false);

    def issueIndexingService = ComponentAccessor.getComponent(IssueIndexingService)

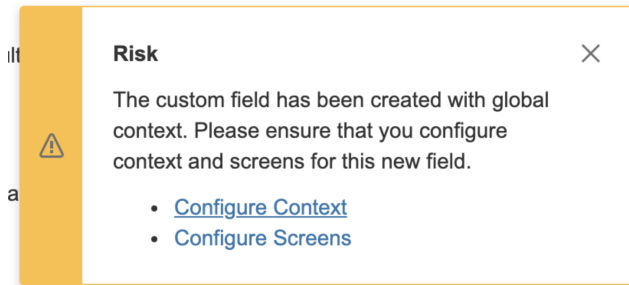
```

```

boolean wasIndexing = ImportUtils.isIndexIssues()
ImportUtils.setIndexIssues(true)
issueIndexingService.reIndex(issueManager.getIssueObject(issue.id))
ImportUtils.setIndexIssues(wasIndexing)
log.debug(value)
}

```

Upon creation, you'll be warned about configuring the context and screen, for choosing to which issue types it's applicable, and to the screens where you want it visible.



Using it

As an example, you may use the previous field as a way to sort and filter Tests in the "Add Tests" dialog, whenever adding them from within an existing Test Execution.

Add Tests to Test Execution CALC-3702
Assignee
Issue Assignee

Select
Search
JQL

Filter(s)

Project
Calculator (CALC)

Test Type

Choose the Test Type

Contains text

Risk Level
L1: Low
L2: Medium
L3: High
L4: Very High
L5: Severe

Clear
Search

Issue Type	Key	Summary	Risk Level	Probability	Impact
<input type="checkbox"/>	CALC-3703	ability to perform clear / start new operations from scratch	L5: Severe	4	4
<input type="checkbox"/>	CALC-1202	CanAddNumbers	L3: High	2	4
<input type="checkbox"/>	CALC-1203	CanDoStuff	L2: Medium	1	4
<input type="checkbox"/>	CALC-3706	check the history of operations	L1: Low	1	2

Showing 1 to 4 of 4 entries
First
Previous
1
Next
Last

Calculating risk at requirement or project level and storing it in a Single Select field at Test level using a Script Listener

This example assumes that we're evaluating the probability and impact (and consequently the risk level) at requirement or at project level (in some specific issue type for that purpose.)

As changes are made to probability or to impact on the parent issue (e.g. requirement, project risk,) a listener will handle the issue updated & created events and will update a "Select List (single choice)" based custom field defined for Test issues. That field needs to be created and configured beforehand will all the possible options.

ISSUE TYPES

[Issue types](#)[Issue type schemes](#)[Sub-tasks](#)

WORKFLOWS

[Workflows](#)[Workflow schemes](#)

SCREENS

[Screens](#)[Screen schemes](#)[Issue type screen schemes](#)

FIELDS

Custom fields[Field configurations](#)

Configure Custom Field: Parent Risk Level

Below are the Custom Field Configuration schemes for this custom field. Scher differently for each project context or in a global context. Moreover, project leve

- [Add new context](#)
- [View Custom Fields](#)

Default Configuration Scheme for Parent Risk Level

Default configuration scheme generated by JIRA

Applicable contexts for scheme: [Edit Configuration](#)

Issue type(s):



Default Value: [Edit Default Value](#)

Options: [Edit Options](#)

- L1: Low
- L2: Medium
- L3: High
- L4: Very High
- L5: Severe

**Please note**

This approach and the code of following script assumes that a Test covers just one issue. If, for example, a given requirement is covered by a Test that also covers another requirement, then whenever editing the impact in one of the requirements will update/overwrite the calculated risk level on the custom field on the Test issue.

Thus, you may have to think if you want to follow this approach as it has some constraints.

Risk (Level) calculated from parent issue using a Script Listener

```
import com.atlassian.jira.issue.fields.CustomField
import com.atlassian.jira.issue.CustomFieldManager
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.event.issue.IssueEvent
import com.atlassian.jira.issue.util.DefaultIssueChangeHolder
import com.atlassian.jira.issue.ModifiedValue
import com.atlassian.jira.issue.history.ChangeItemBean
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.issue.customfields.option.Option
import com.atlassian.jira.event.type.EventDispatchOption
import com.atlassian.jira.issue.index.IssueIndexingService
import com.atlassian.jira.util.ImportUtils
import com.atlassian.jira.bc.issue.search.SearchService
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter
import org.apache.log4j.Logger
import org.apache.log4j.Level

serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()

Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    def searchService = ComponentAccessor.getComponent(SearchService.class);
    SearchService.ParseResult parseResult = searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
    }
}
```

```

        searchResults = results.getIssues();
        return searchResults;
    }

    return []
}

def log = Logger.getLogger("com.onresolve.jira.groovy")
log.setLevel(Level.DEBUG)

MutableIssue issue = event.issue
def issueType = issue.issueTypeObject.name

// Only process Story and Risk issues
if (issueType != "Story" && issueType != "Risk"){
    return
}

String cfImpactName = "Impact"
String cfProbabilityName = "Probability"
String cfRiskLevelName = "Parent Risk Level"

def changeManager = ComponentAccessor.getChangeHistoryManager();
def changeLog = event.getChangeLog();

def impactChanged = false, probabilityChanged = false;

if ( changeLog != null ) {
    def changeId = changeLog.getLong( "id" );
    if ( changeId != null ) {
        def change = changeManager.getChangeHistoryById( changeId );
        for (ChangeItemBean bean : change.getChangeItemBeans() ) {
            if ( bean.getField().equals(cfImpactName)
                && ChangeItemBean.CUSTOM_FIELD.equals( bean.getFieldType() ) ) {
                log.debug(cfImpactName + " changed");
                impactChanged = true;
            }
            if ( bean.getField().equals(cfProbabilityName)
                && ChangeItemBean.CUSTOM_FIELD.equals( bean.getFieldType() ) ) {
                log.debug(cfProbabilityName + " changed");
                probabilityChanged = true;
            }
        }
    }
}

if (impactChanged || probabilityChanged){
    def issueManager = ComponentAccessor.issueManager
    def customFieldManager = ComponentAccessor.customFieldManager
    def cfImpact = customFieldManager.getCustomFieldObjectByName(cfImpactName)
    def cfProbability = customFieldManager.getCustomFieldObjectByName(cfProbabilityName)

    def option

    def impact
    def probability
    option = (Option)issue.getCustomFieldValue(cfImpact)
    if (option != null) impact = option.getValue()
    option = (Option)issue.getCustomFieldValue(cfProbability)
    if (option != null) probability = option.getValue()

    def risk
    def level

    if ((impact) && (probability)) {
        risk = Double.parseDouble(impact) * Double.parseDouble(probability)
    }
}

```

```

    } else {
        risk = null
    }

    if ((risk>0) && (risk<4)) {
        level = "L1: Low"
    } else if ((risk>=4) && (risk<8)) {
        level = "L2: Medium"
    } else if ((risk>=8) && (risk<12)) {
        level = "L3: High"
    } else if ((risk>=12) && (risk<16)) {
        level = "L4: Very High"
    } else if (risk==16) {
        level = "L5: Severe"
    } else {
        level = null
    }

    def cfRiskLevel = customFieldManager.getCustomFieldObjectByName(cfRiskLevelName)

    def jql = "issue in requirementTests('${issue.key}')"
    def testIssues = getIssues(jql)
    testIssues.each {
        MutableIssue testIssue = issueManager.getIssueObject(it.key)
        def fieldConfig = cfRiskLevel.getRelevantConfig(it)
        def value = ComponentAccessor.optionsManager.getOptions(fieldConfig)?.find { it.toString() == level }
        testIssue.setCustomFieldValue(cfRiskLevel, value)
        issueManager.updateIssue(serviceAccount, testIssue, EventDispatchOption.DO_NOT_DISPATCH, false);

        def issueIndexingService = ComponentAccessor.getComponent(IssueIndexingService)
        boolean wasIndexing = ImportUtils.isIndexIssues()
        ImportUtils.setIndexIssues(true)
        issueIndexingService.reIndex(issueManager.getIssueObject(testIssue.id))
        ImportUtils.setIndexIssues(wasIndexing)
    }

    log.debug(value)
}

```

Other non-standard configuration approaches

These configuration approaches have several drawbacks, specially in terms of usability; thus, they are not recommended. They are provided here for reference though, in case you still want to implement the fields this way.

Calculating risk as a number using a Script Field

In this example, we're calculating the risk as a number using a custom Script Field (**Add-ons > ScriptRunner > Script Fields > Add New Item > Custom Script Field**).

Script Fields

A script field is a custom field that allows you to automatically display a value according to the results of a ScriptRunner script. Use this page to create your script fields.

Looking for more information?

Check out our [documentation for Script Fields](#). If you have a question, the [Atlassian Community](#) may also have answers for you!

Add New Item ↗

➤ **Custom Script Field** ?

Create your own custom scripted field.

➤ **Issue(s) picker** ?

Select another issue, optionally constrained by a JQL query

➤ **Date of First Transition** ?

Shows the date when the field first transitioned into :

➤ **No. of Times In Status** ?

This field is calibrated to show the number of times :
been in selected status

Custom Script Field ?

Create your own custom scripted field.

Field Name

Risk

The name of the custom field that you are creating

Field Description

Risk Level

The description of the custom field that you are creating

Note

Risk Level (RBL)

An optional note, useful only for your reference.

Template

Number Field

Template to represent your field. Match at the output of your script.

Script file

Start typing to search for files...

Point to the script accessible on the server

Inline script

```
1 import com.atlassian.jira.issue.Issue
2 import com.atlassian.jira.issue.MutableIssue
3 import com.atlassian.jira.issue.IssueManager
4 import com.atlassian.jira.component.ComponentAccessor
5 import org.apache.log4j.Logger
6 import org.apache.log4j.Level
7
8
9 
```

Enter your script here

Risk (Level) calculated as a number

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.component.ComponentAccessor
import org.apache.log4j.Logger
import org.apache.log4j.Level

def log = Logger.getLogger("com.onresolve.jira.groovy")
log.setLevel(Level.DEBUG)

def issueManager = ComponentAccessor.issueManager

def impact = getCustomFieldValue("Impact").toString()
def probability = getCustomFieldValue("Probability").toString()

def risk

if ((impact!="null") && (probability!="null")) {
    risk = Double.parseDouble(impact) * Double.parseDouble(probability)
} else {
    risk = null
}

return risk
```



Please note

Even if risk value related custom field is created using the Template "Number Field", as shown earlier, sorting won't work as expected as it will be done by text comparison (i.e. alphabetically).

Columns

<input type="checkbox"/>	Issue Type	Key	Summary	Test Type	Risk
<input type="checkbox"/>		CALC-1202	CanAddNumbers	Generic	6.0
<input type="checkbox"/>		CALC-1203	CanDoStuff	Generic	16.0

Showing 1 to 2 of 2 entries

First Previous 1 Next

You will need to guarantee that Jira uses the "Number Searcher" by editing the custom field details. If you change this, don't forget to re-index.

Edit Custom Field Details

If the search template is changed, manual reindexing must follow

Field Name

Description

A description of this particular custom field.
You can include HTML, make sure to close all your tags.

Search Template

Note that changing a custom field searcher may require a [re-index](#).

Calculating risk as text using a Script Field

In this example, we're calculating the risk level and presenting it in a friendly/textual way. For this, we'll use a custom Script Field (**Add-ons > ScriptRunner > Script Fields > Add New Item > Custom Script Field**).

This approach has one side effect though: it will harden the search of issues by value as you have to write the value to search for, whereas a Select List based field can easily be searched by picking the possible values you want to filter.

Risk (Level) calculated and presented as text

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.component.ComponentAccessor
import org.apache.log4j.Logger
import org.apache.log4j.Level

def log = Logger.getLogger("com.onresolve.jira.groovy")
log.setLevel(Level.DEBUG)

def issueManager = ComponentAccessor.issueManager

def impact = getCustomFieldValue("Impact").toString()
def probability = getCustomFieldValue("Probability").toString()

def risk
```

```
def level
  if ((impact) && (probability)) {
    risk = Double.parseDouble(impact) * Double.parseDouble(probability)
  } else {
    risk = null
  }

  if ((risk>0) && (risk<4)) {
    level = "L1: Low"
  } else if ((risk>=4) && (risk<8)) {
    level = "L2: Medium"
  } else if ((risk>=8) && (risk<12)) {
    level = "L3: High"
  } else if ((risk>=12) && (risk<16)) {
    level = "L4: Very High"
  } else if (risk==16) {
    level = "L5: Severe"
  } else {
    level = null
  }

  return level
end def
```

References

- Risk management - Principles and guidelines: ISO 31000:2009(E)
- Souza, Ellen Polliana & Gusmao, Cristine & Venâncio, Júlio. (2010). Risk-Based Testing: A Case Study. 1032 - 1037. 10.1109/ITNG.2010.203.
- RISK FACTORS IN SOFTWARE DEVELOPMENT PHASES, Haneen Hijazi, Msc Hashemite University, Jordan Shihadeh Alqrainy, PhD; Hasan Muaidi, PhD; Thair Khmour, PhD; Albalqa Applied University, Jordan