ATLASSIAN University

Getting More from Jira Workflows

1 Course Overview

What will you learn?

- Use advanced workflow features
- Deal with complex workflow requirements
- Learn best practices for creating and managing workflows

In this course you'll learn how to use advanced workflow features – conditions, validators, post functions, triggers, properties, etc. – to build awesome workflows. You'll also learn how to deal with complex workflow requirements and learn best practices for creating and managing workflows. And a lot more...

To succeed here, you need to have

- Familiarity with Jira administration and administering workflows
- Optional courses:
  - Jira Administration Part 1
  - Jira Administration Part 2

You should be familiar with Jira administration and administering workflows (alternatively having attended both Atlassian University Jira Administration courses). You'll get most from the class if you have knowledge of the following:

- How to manage users, groups, roles, and permissions
- Familiarity with general Jira functionality and terminology (screens, fields, projects, issues, schemes)
- How to create and use the workflow elements – statuses and transitions
- Familiarity with workflow terminology such as workflow and workflow scheme

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

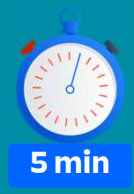
Triggering Transitions

Extending Workflows with Properties

Taking It to the Next Level

Lab 1 – Starting Your Lab Environment

- Use the link provided to access your lab VM
- Start your Jira instance



You won't be doing anything in Jira yet, but you need to start up your lab VM so it's available for the next lab.

2 Covering the Basics

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

Triggering Transitions

Extending Workflows with Properties

Taking It to the Next Level

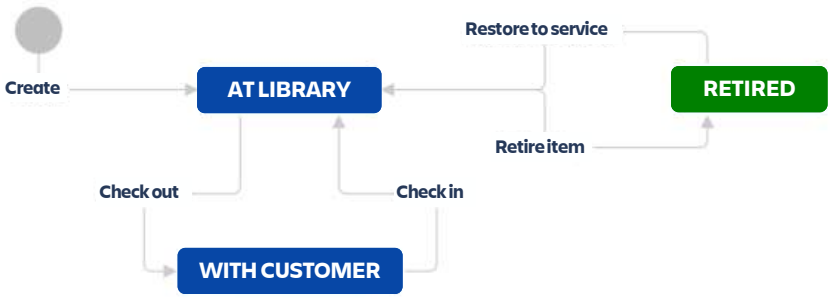
What will you learn?

- Review conditions, validators, post functions, triggers, and properties
- Configure boards
- Identify who can do the different workflow tasks
- Change the workflows used by different issue types in projects

Most of the material in this module will likely be review for you. We'll be going over the basics to ensure you're ready for the material in the rest of the course. Here you'll learn how to:

- List the purpose of conditions, validators, post functions, triggers, and properties
- Configure boards to reflect changes you make in the underlying workflow
- Identify the tasks different types of users (project admins, board admins, and Jira admins) can do with workflows
- Change the workflows used by different issue types in projects by using workflow schemes
- And more...

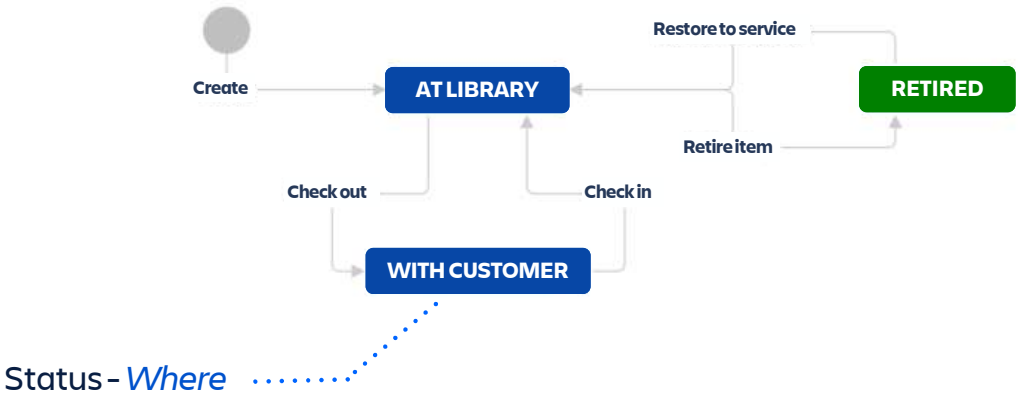
What's a Workflow?



Workflow defines the process for completing the tasks – what actions need to occur, when and by who. It's a business process, with stages that an issue goes through to reach its final status. Your Jira issues can follow a process that mirrors your team's practices. Each issue has a workflow and you can define different workflows for different issue types. Workflow is the backbone of how Jira works within your organization; taking the time to customize the workflow to meet your business needs will pay many dividends down the line.

Here you see an example workflow. Libraries have well-known workflows surrounding materials they lend out to the public. Each item (e.g. book, magazine, etc.) could be stored in Jira as an issue and administered in the workflow shown here.

What's a Status?

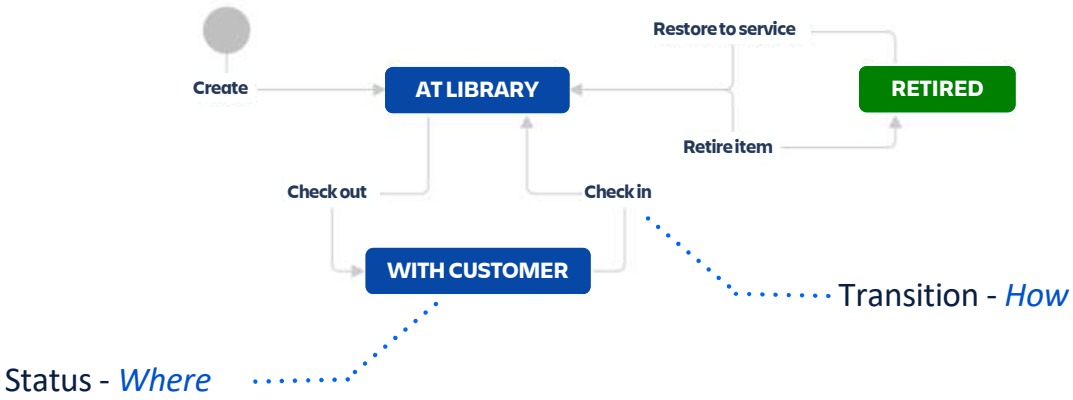


A status represents the state of an issue at a particular point in a specific workflow. In the library example, a book can be in one of three places: at the library, with a customer, or retired from inventory. An issue can be in only one status at a given time. Every status should be a unique state in your workflow. Statuses must describe what has already happened.

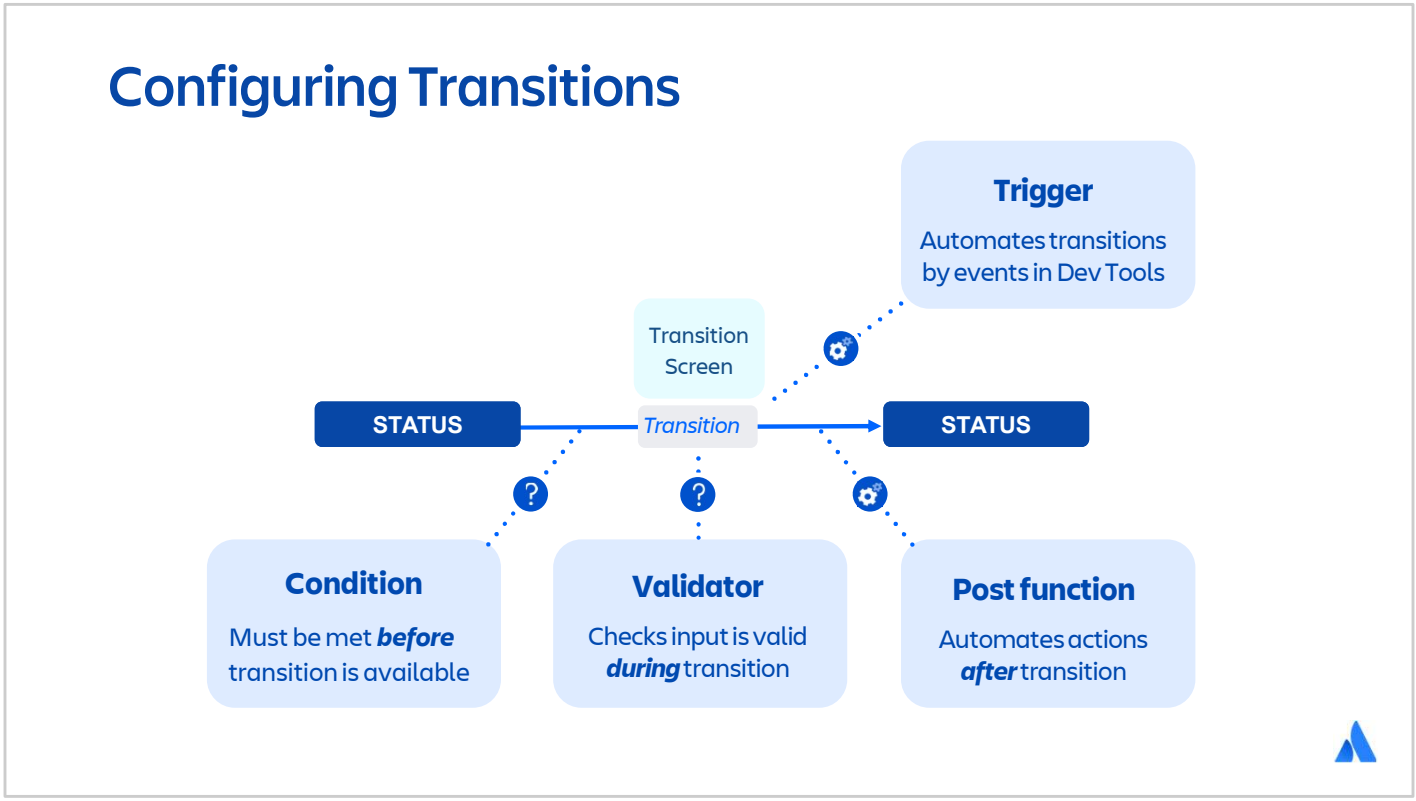
Jira ships with a set of default statuses that are used by the default workflows such as Open, In Progress, Reopened, Resolved, Closed, To Do, Done, etc. When editing workflows, you can use the default statuses that Jira ships with or create your own (if you're a Jira admin).

Don't confuse resolution with status – Status describes where an item is in the workflow; resolution explains why an item is not in flight anymore. To retire a book from circulation, our librarian should create a status called retired, and options for resolution like lost, damaged, out of date, etc. Searching for everything that's retired (or resolved) – with the ability to search for why it's retired – gives much better reporting metrics. That's why Jira uses resolution - to know if the issue is considered inactive by the organization.

What's a Transition?



Transitions are the bridges between statuses; the way a particular issue moves from status to status. At the library, books are loaned out by librarians, returned by customers, and evaluated by library staff to see if they're fit for circulation. Note that customers cannot evaluate if an item is fit for inventory – that must be decided by the library staff. Jira is able to set workflow permissions that allow only the right people to transition an issue. You can also, optionally, use screens to gather data/information during transitions. A transition is a link between two statuses that enables an issue to move from one status to another. In order for an issue to move between two statuses, a transition must exist. A transition is a one-way link, so if an issue needs to move back and forth between two statuses, two transitions need to be created.



You can configure your workflow transitions with conditions, validators, post functions, and triggers. These usually map to either a convenience (do the boring stuff for me) or a requirement (e.g. you can't close that without adding a fix version.)

A condition specifies a situation that must exist BEFORE something else is permitted (pre-requisite). Conditions control if a transition is available. They can take into consideration a user's project roles and permissions, the status of any subtasks, as well as the state of source code associated with issues.

A validator checks that any input made DURING the transition is valid before transition is performed. If a transition's validator fails, the transition's post functions are not executed. When this happens the issue does not progress to the destination status of the transition.

A post function is an automated action AFTER a transition has been executed. Post functions carry out any additional processing required after the transition, such as updating an issue's fields, generating change history, adding a comment, and generating an event to trigger email notifications.

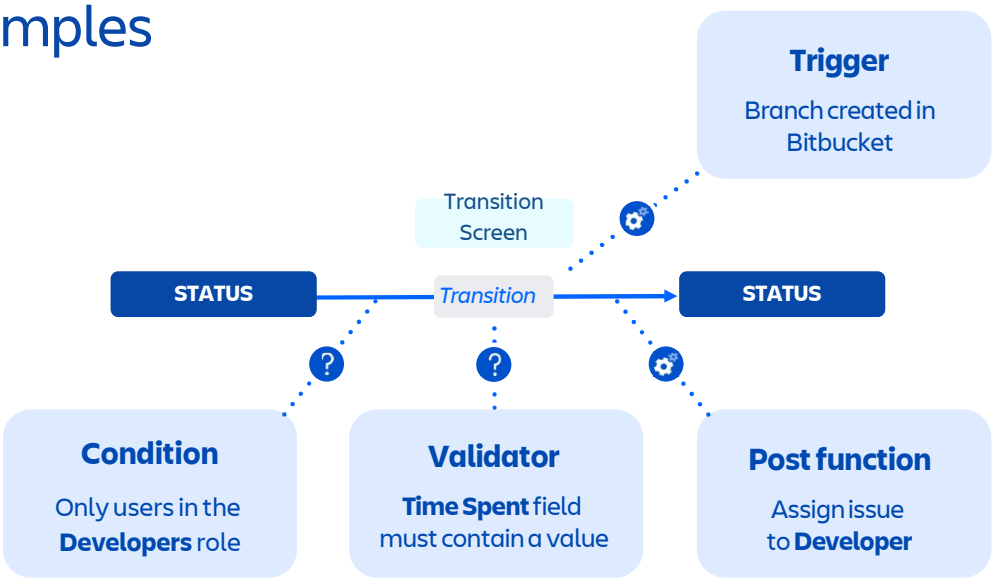
Triggers automatically transition issues when certain events occur in your development tools such as Bitbucket.

You can also add transition screens to transitions to gather required data.

See examples of all of these on the next slide.

We'll cover conditions and validators in the next module, post functions in module 4, and triggers in module 5.

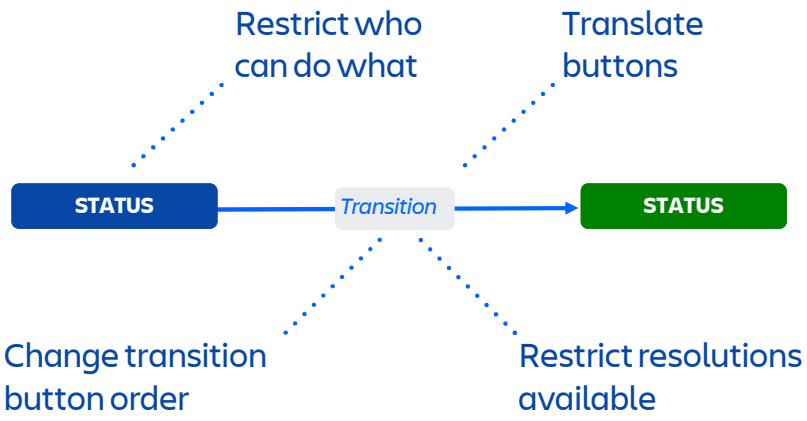
Configuring Transitions Examples



Here are some examples of the way you can configure transitions:

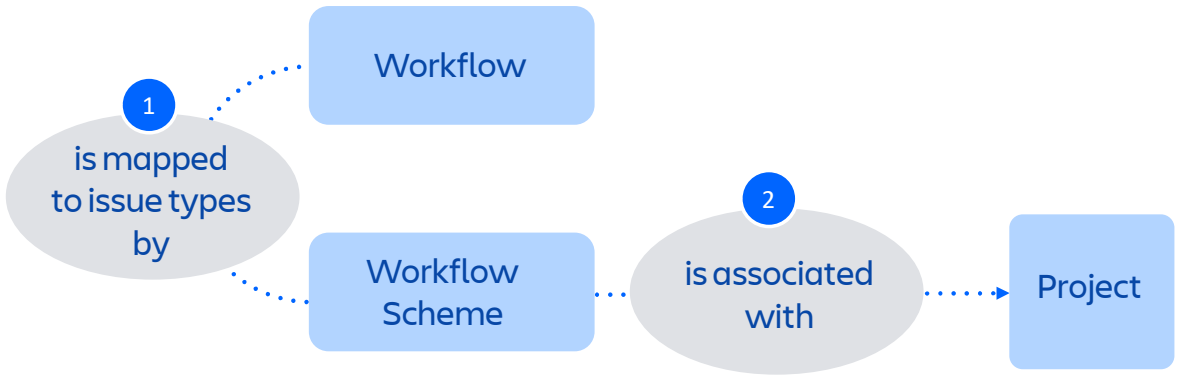
- Condition - only users in the Developers role will see the Code Review transition. If a condition fails, the user will not see the transition link when viewing the issue.
- Validator - the Time Spent field must contain a value. If a transition's validator fails, the transition's post functions are not executed. When this happens the issue does not progress to the destination status of the transition.
- Post function - assigning an issue back to the user specified in the Developer field when code review fails.
- Triggers - if a branch is created in Bitbucket an open issue can be automatically transitioned into the In Progress status.
- Transition screen – gets a comment when issues fail code review and are transferred back to the Developer.

Enriching Your Workflow with Properties



Properties are a hidden gem of Jira. You can use them to further customize and enrich your workflow. For example, you can implement restrictions on who can do what in a certain status. Or restrict what resolutions are available in a transition. Or even reorder the transition buttons within issues or translate buttons. We'll cover properties in module 6.

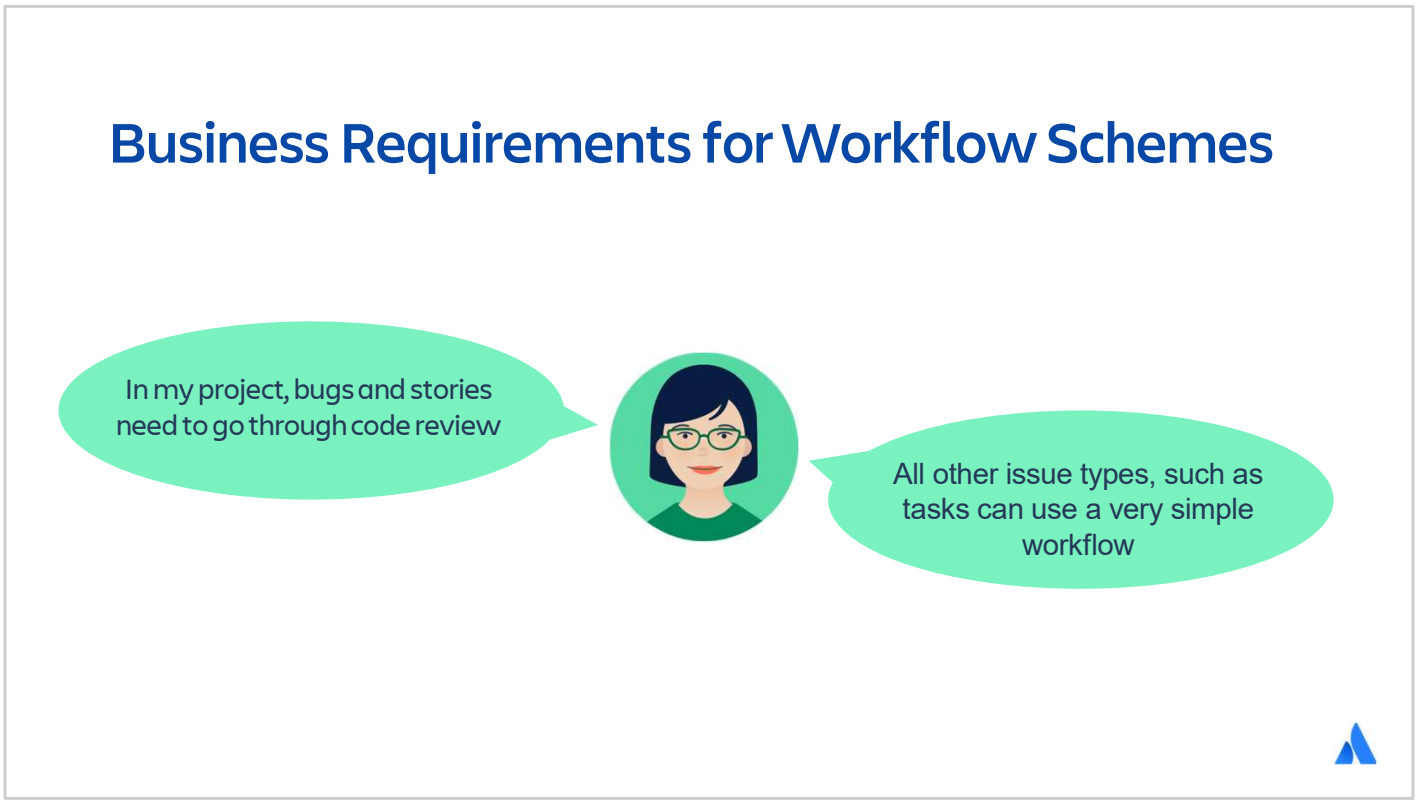
Workflow Schemes



Once you've completed your workflow, you associate with issue types in a workflow scheme. Workflow schemes are a defined set of associations – or mappings – between workflows and issue types. Then you associate your workflow scheme with a project (or projects). This makes it possible to use a different workflow for every combination of project and issue type.

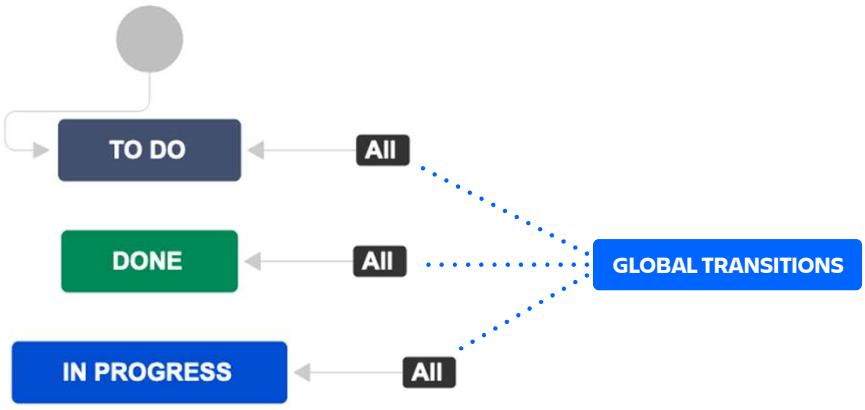
You can associate a single workflow scheme with more than one project but only one workflow scheme can be associated with a given project.

The issue type scheme associated with a project defines the issue types that are available to that project. If an issue type is not defined in the project's issue type scheme, its workflow is not used.



In this example a workflow scheme is used to specify that issues of type Bug and Story will go through a custom workflow that includes code review. Whereas other issue types such as Task, Sub-task, etc. can go through a very simple workflow such as the Software simplified workflow.

The Simplified Workflow

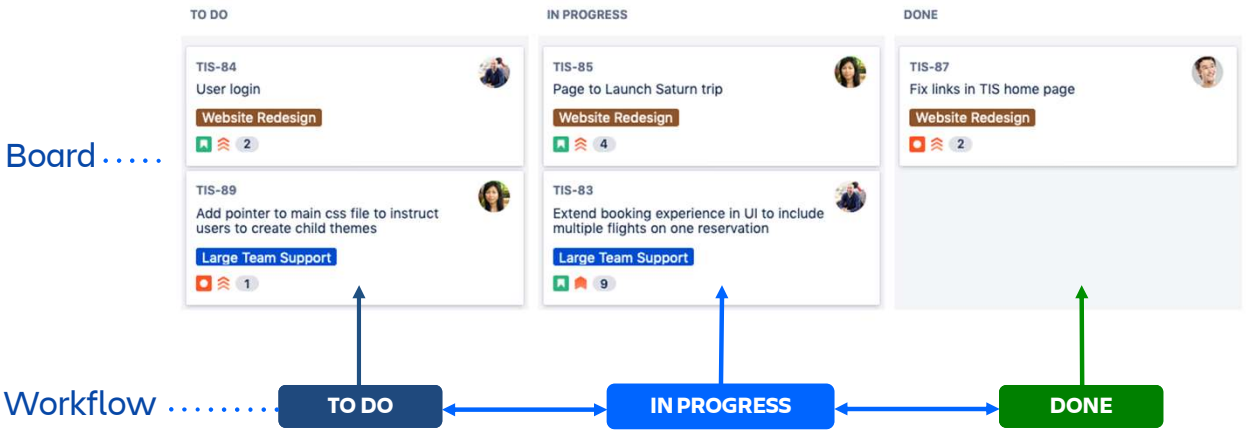


The simplified workflow is very basic and all its transitions are global transitions i.e. to ALL. Global transitions allow any status in a workflow to transition to a particular status. Here you see the default workflow for a Scrum software development project. It has only three steps and issues can freely move between any statuses.
The simplified workflow can only be used if a board represents a single project.
See https://confluence.atlassian.com/jirasoftwareserver/using-the-simplified-workflow-938845286.html.
Also note the different colors of each status. Each of these represents a status category – To Do (blue), In Progress (yellow), and Done (green). You'll see these colors in all types of workflows. These help you identify where issues are in their lifecycle, particularly in places where a large number of issues are rolled up e.g. Version Details page and Sprint Health gadget.

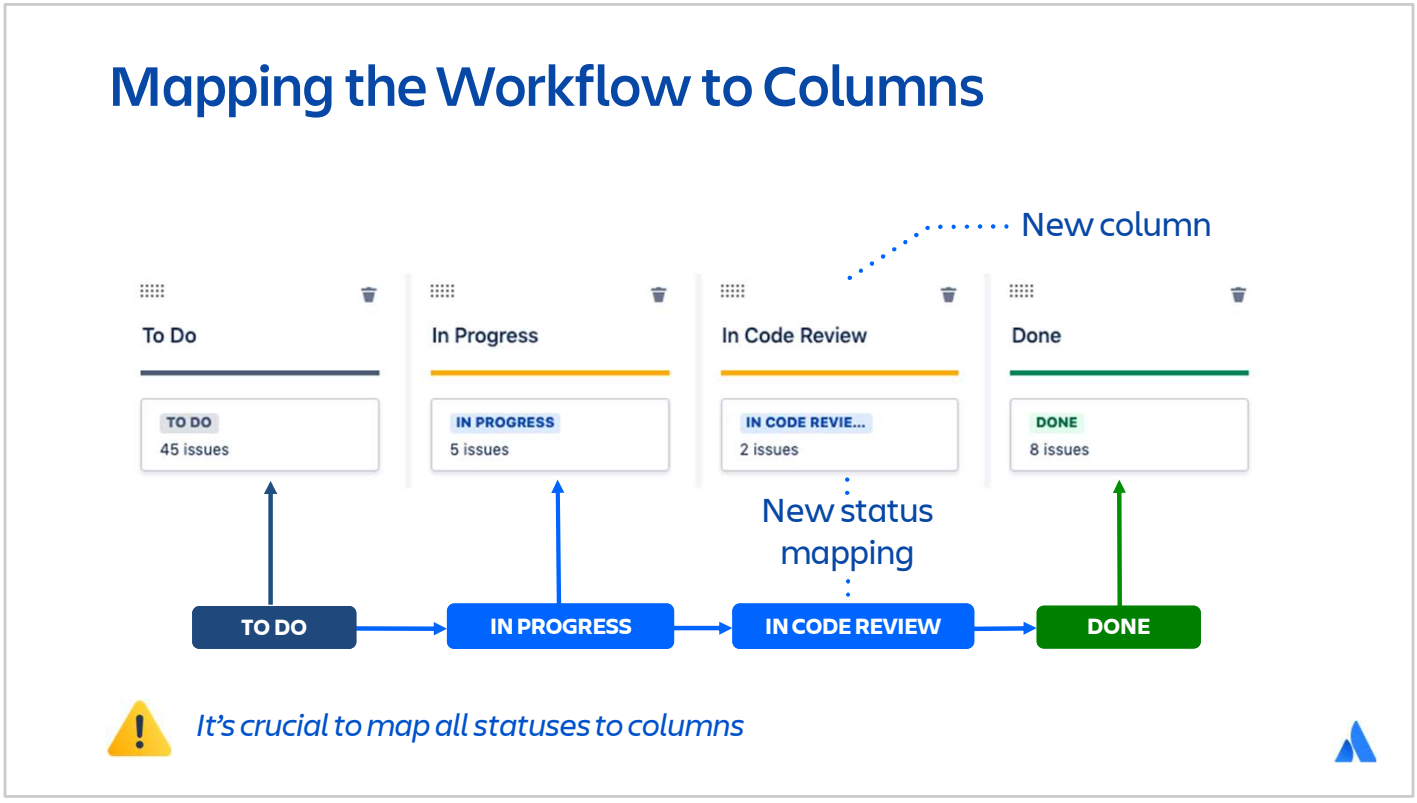
SIMPLIFIED	CUSTOM / JIRA
A simple workflow	A more complex workflow
Drag issues freely on the board	Specific sequence of steps
Easy to use	Conditions, validators, and automated functions
No transition screens	Custom transition screens for input
Board and project admins can edit	Jira admins can edit, project admins have limited editing

The simplified workflow is an easy to use workflow. There are no screens for transitions and users can transition an issue from any status to any other status (drag freely between columns). Also users who are both board and project admins can edit the workflow via the board configuration, adding statuses and columns.
When you use the default Jira workflow, or a workflow customized from the default Jira workflow you get a more complex workflow. Screens pop up during transitions that allow users to enter data for example, a resolution. Also you can add conditions, user input validation and automated functions to transitions in the workflow. It has a specific sequence of steps rather than allowing every status to transition into every other status. And lastly only Jira admins can fully edit the workflow so you have more control. Project admins have limited editing capability for their project workflows. More on that soon.
To be a Jira admin you need the 'Jira Administrators' global permission.

Boards & Underlying Workflow



Boards are tied to the underlying workflow. For Software projects, the simplified workflow is used for the board that's created when the project is created.
Here's an example of a simplified workflow for a scrum software development project. The top screenshot shows the sprint board in the project. The diagram at the bottom shows the underlying workflow. Note the connection between columns and workflow statuses. This is the actual workflow that's used for the board.



Here you see some of the things that can be changed with regard to the relationship between workflow, status, and columns of the board. You can create a new column on the board and then map a status to that column. You can also map a status to an existing column. It's crucial that you map all statuses to columns. Otherwise issues in "unmapped" statuses will be hidden from the board.

You can add columns to boards that are using the simplified workflow or any other type of workflow. But to add a status to a board the project must be using the Simplified Workflow. Adding a status to a board adds a status to the underlying workflow.

Configuring Boards

Jira admin	Project admin + Board admin	Board admin only	Project admin only	
✓	✓	✓		Add columns to a board and map statuses using any type of workflow
✓	✓			Add new statuses to a board using the simplified workflow
			✓	Cannot configure the board

Be watchful of users creating a lot of statuses

You can add columns to the board if you're a Jira admin or a board admin.
Jira admins can add statuses to boards. Also you can add statuses if you're the project admin for the one project that is on the board as well as being a board admin for the board. In other words, you need to be either a Jira admin OR a project admin and a board admin to configure the board and add both statuses and columns to the simplified workflow for the project. If you are only the board admin (and not a Jira admin or the project admin) you can configure the board but you can only add columns to the board but not statuses to the simplified workflow. If you are only the project admin (and not a Jira admin or the board admin) you cannot add statuses or columns to the board.
So users who are project admins and board admins can work with their project's simplified workflow and not get a Jira admin involved. But be watchful, as you can end up with a lot of statuses if users can freely add their own statuses to their boards in their projects!

ACTIVE	INACTIVE
Currently being used	Not in use
Associated with an active workflow scheme and a project	Not associated with any workflow scheme or project
Can't be edited – edit draft	Can be edited

There are differences between editing an inactive and an active workflow. There are restrictions on the modifications you can make to an active workflow, due to the impact the changes will have on projects and/or issue types that use this workflow.

- An inactive workflow is a workflow that is not currently being used by any projects. Because there are no issues currently transitioning through an inactive workflow, you can edit the workflow's steps and transitions directly.
- An active workflow is a workflow that is currently being used by one or more projects (in an active workflow scheme). When you edit an active workflow, Jira first creates a draft of it, that you can then modify. When you've finished, you can publish your draft and, optionally, save your original workflow as an inactive backup.

The following limitations apply when editing the draft for an active workflow:

- It is not possible to edit the workflow name (only the description) if a workflow is active.
- Workflow statuses cannot be deleted.
- The step ID cannot be changed.

To make any of the modifications listed above, you need to copy the workflow, modify the copy and then activate it. We'll cover this in more detail later in the course.

Workflows need to be activated to use them in Jira. Activating is the process of mapping the workflow to a workflow scheme, and then associating the workflow scheme with a project.

PROJECT ADMINS CAN	PROJECT ADMINS CAN'T
• Edit the workflow associated with their project • Create, update, delete transitions • Add existing statuses to their workflow • Delete statuses that aren't used by their project's issues	• Edit shared workflows • Edit the jira default system workflow • Select or update a transition screen • Edit transition properties, conditions, validators, or post functions

Project admins can edit their project's workflow as long as the workflow:

- Is not shared with any other projects
- Is not the Jira default system workflow, which cannot be edited at all

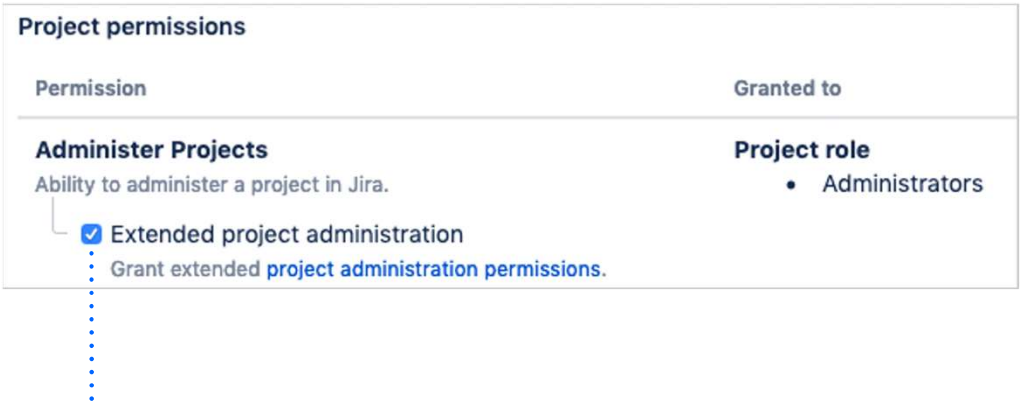
Project admins can't edit the workflow to the same extent as a Jira admin. The restrictions are:

- To add a status, the status must already exist in the Jira instance i.e. the project admin can't create new statuses or edit existing statuses.
- To delete a status, the status must not be used by any of the project's issues.
- The project admin can create, update (name and description) or delete transitions, but they can't select or update a screen used by the transition, or edit or view a transition's properties, conditions, validators or post-functions.

See https://support.atlassian.com/jira-software-cloud/docs/use-the-simplified-workflow/ (Cloud) or https://confluence.atlassian.com/jirasoftwareserver/using-the-simplified-workflow-938845286.html (DC/Server).

If you're upgrading from an earlier version of Jira, Atlassian provides scripts that will help Jira admins work out what the impact to their instance will be. The scripts return a list of your projects, and which groups and/or users will be able to edit the project's workflows. There's also a new 'workflow' event to the audit log that'll let you know who made workflow changes, and to which projects. See the release notes for where to find the scripts.

Restricting Project Admins



De-select to disable project admins from editing their workflow

In Jira Data Center/Server, Project admins can edit workflows (and screens) in their projects by default. This is controlled by an option for the Administer Projects permission called 'Extended project administration'. If you're a Jira admin, you can disable it by editing the relevant permission scheme and deselecting the option.

See how it's done

- Exploring workflow schemes and workflow in Jira administration
- Editing workflow

Instructor demo

Are you getting it?

In what order will post functions, conditions and validators occur in a transition?

- Validator
- Post function
- Condition

The answer is on the next slide.

Did you get it?

During a transition they will occur in the following order:

1. Condition
2. Validator
3. Post function

Answer: During a transition conditions must exist before the transition is permitted, input made during the transition must be valid (validator), and automated actions (post functions) occur after the transition.

Are you getting it?

A workflow is mapped to issue types by a _____ which is then associated with a project.

a. Transition
b. Issue type scheme
c. Workflow scheme
d. Status

A workflow is mapped to issue types by a _____ which is then associated with a project.

a. Transition

b. Issue type scheme

c. Workflow scheme

d. Status

Did you get it?

A workflow is mapped to issue types by a **workflow scheme** which is then associated with a **project**.

a. Transition
b. Issue type scheme
✓ c. Workflow scheme
d. Status

Answer: A workflow is mapped to issue types by a workflow scheme which is then associated with a project.

Are you getting it?

Given these requirements would it be better to use a **simplified workflow** or a **custom workflow**?

- It should be very easy to use
- Users should be able to drag issues anywhere on the board
- There should be no transition screens
- Board and project administrations should be able to configure the board

Given these requirements would it be better to use a **simplified workflow** or a **custom workflow**?

- It should be very easy to use
- Users should be able to drag issues anywhere on the board
- There should be no transition screens
- Board and project administrations should be able to configure the board

Did you get it?

A **simplified workflow** meets the following requirements:

a. It should be very easy to use
b. Users should be able to drag issues anywhere on the board
c. There should be no transition screens
d. Board and project administrations should be able to configure the board

Answer: A simplified workflow.

Are you getting it?

Which of these can project admins do to the workflow associated with their project (that hasn't been shared with another project)?

a. Delete a transition
b. Delete a status that's used by their project's issues
c. Add an existing status
d. Select a transition screen
e. Edit transition conditions and validators

This assumes the project admin is not a Jira admin.
The answer is on the next slide.

Did you get it?

Which of these can project admins do to the workflow associated with their project (that hasn't been shared with another project)?

- ✓ a. Delete a transition
- b. Delete a status that's used by their project's issues
- ✓ c. Add an existing status
- d. Select a transition screen
- e. Edit transition conditions and validators

Answer: Project admins can delete a transition (as well as create and update name and description) and add an existing status to their workflow. They cannot delete a status if it's being used by any of their project's issues. Nor can they select (or update) a transition screen. They also cannot edit transition conditions and validators (nor properties or post functions or triggers).

Takeaways

- You need to be a Jira admin to fully edit a workflow
- Ensure you map all statuses to columns on your board
- Be watchful of users creating a lot of statuses
- Associate workflow schemes to projects make workflows active

Project admins only have limited editing capabilities for workflows. To fully edit a workflow and add conditions, validators, post functions, etc. you need to be a Jira admin. It's crucial that you map all statuses to columns. Otherwise issues in "unmapped" statuses will be hidden from the board.
Users who are project admins and board admins can work with their project's simplified workflow and not get a Jira admin involved. But be watchful, as you can end up with a lot of statuses if users can freely add their own statuses to their boards in their projects!
You need to associate workflows with issue types in a workflow scheme then associate that workflow scheme to a project for workflows to become active.

Lab 2 – Covering the Basics

- Exercise 1:
  - Explore a workflow and associated board configuration as a project admin
  - Rename a workflow status as a Jira admin
  - Configure a board
- Exercise 2:
  - View a simplified workflow
  - Add a second workflow to a workflow scheme

3 Creating Conditions & Validators

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

Triggering Transitions

Extending Workflows with Properties

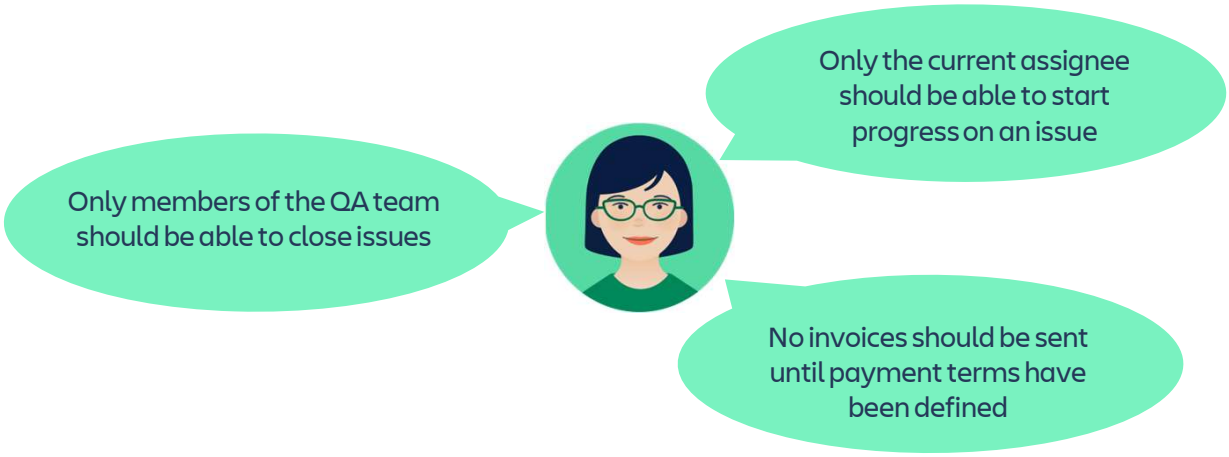
Taking It to the Next Level

What will you learn?

- Control who can execute transitions by creating conditions
- Ensure you get the right data during transitions by creating validators

Here you'll learn how to control who can execute transitions by creating conditions. You'll also learn how to ensure you get the right data during transitions by creating validators.

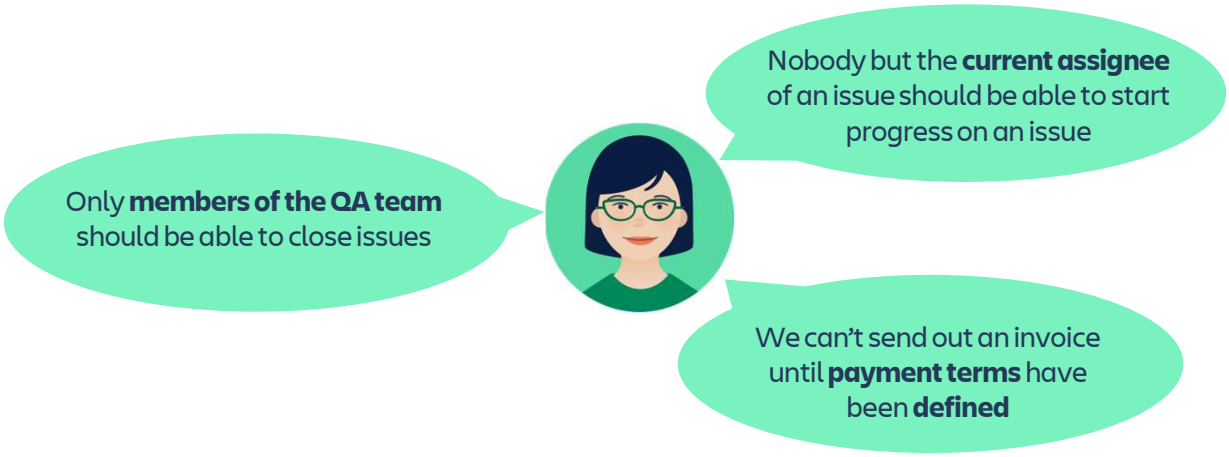
Common Requirements



What do these requirements have in common?

See the next slide for the answer.

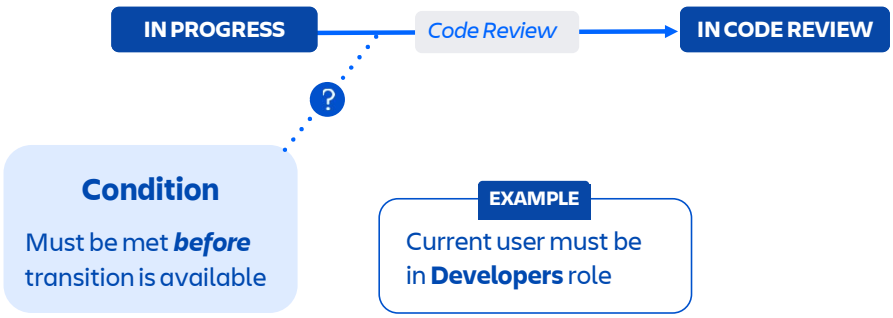
Common Requirements



They're all situations that must exist before a transition can be executed

Answer: All these requirements are situations that must exist before a transition can be executed. Conditions are pre-requisites for the transition.
The user has to be a member of the QA team to close issues.
The user has to be the current assignee to start progress on the issue.
The payments terms must have been defined to send out an invoice.

Conditions



If a condition fails, the user will not see the transition link in the issue

A condition specifies a situation that must exist BEFORE something else is possible or permitted. It's a pre-requisite and controls if a transition is available (before it can be executed). Conditions control either who performs a transition or under what circumstances. In this example, Teams in Space want just developers to be able to send an issue for code review. This can be achieved by adding a condition to the "Code Review" transition. So only the users who are in the Development role can see this transition.

If a condition fails, the user will not see the transition button when viewing the issue, and so will not be able to execute the transition. Conditions cannot validate input parameters gathered from the user on the transition's screen – you need a validator to do this (we'll cover these next).

Common Conditions

WHO

- Assignee
- Reporter
- In project role
- In group
- Has permission

WHAT

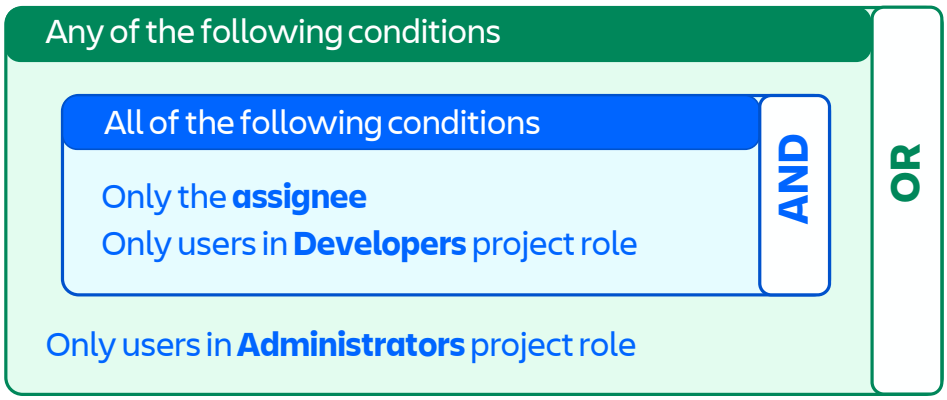
- Code has been committed
- Status of sub-tasks

The most common use cases involve *either* the status of the current user (is assignee or reporter, in roles or groups, has permission, etc.) i.e. the WHO *or* the current state of the issue (code has been committed sub-tasks have been closed, etc.) i.e. the WHAT.

Conditions can take into consideration:

- Who the user is. For example, are they the reporter or the assignee? Are they in a particular project role or in a group. An example of this is restricting an approval transition to only users who are the approvers role.
- A user's permissions, for example, if a user has the Approve permission.
- The status of any subtasks. This is useful when you need to restrict a transition to only happen once all sub tasks associated with an issue have been closed. An example is restricting an issue from being closed until sub tasks associated with it have been closed.
- The state of source code associated with issues, for example if code has, or has not, been committed against this issue.

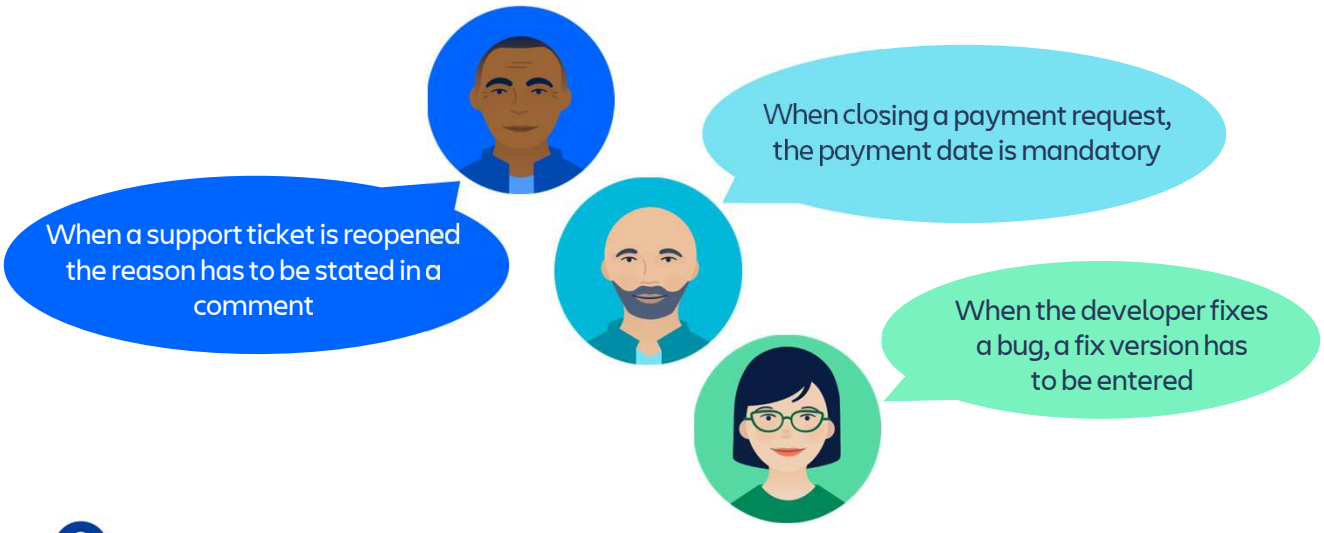
Grouping & Nesting Conditions



Who can execute this transition?

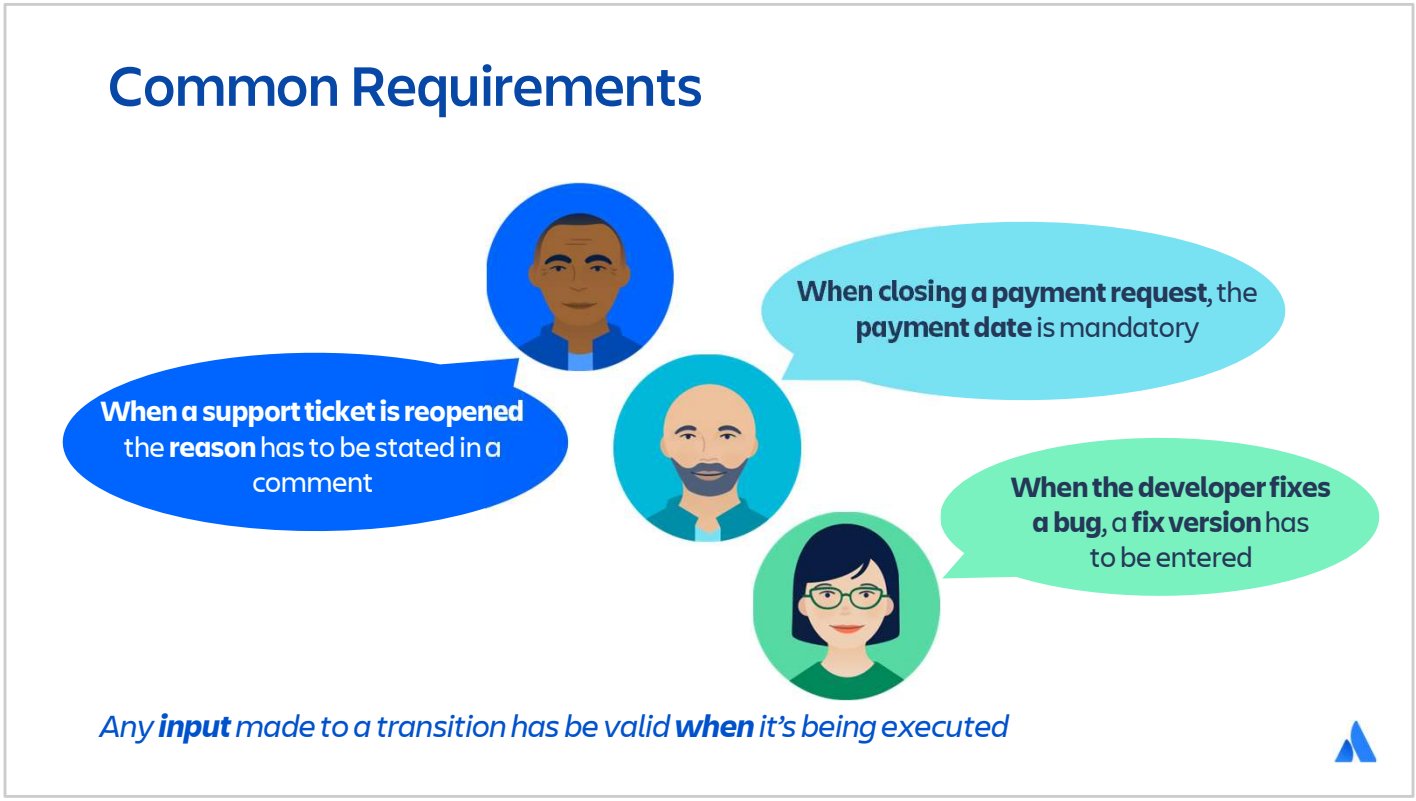
You can also construct complex conditions by grouping and nesting conditions. And you can toggle the logic for how the conditions in a group are applied between All and Any. In any group either all the conditions have to be met or any of the conditions have to be met.
In this example the only users who can execute this condition are either the current assignee who's also in the Developers project role or the project administrator.

Common Requirements



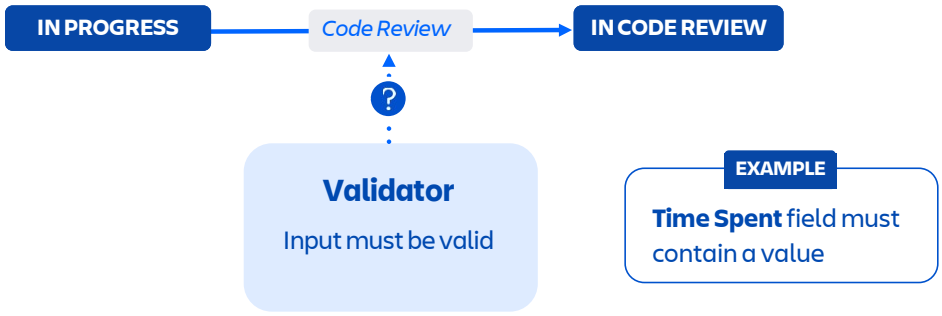
What do these requirements have in common?

See the next slide for the answer.



Answer: All these requirement define input that must be made during the transition.

Validators



If a validator fails, the issue doesn't progress to the next status, and the post functions aren't executed

At Teams in Space, as soon as one of the developers fixes the bug (resolves the issue) they need to ensure the Time Spent field has a value so reports can be run on it. Validators check that any input made to the transition is valid before the transition is performed. Input can include that gathered from the user on the transition's screen. If a validator fails, the issue does not progress to the destination status of the transition, and the transition's post functions are not executed.

Validators check that any input made to the transition is valid before the transition is performed. Input can include that gathered from the user on the transition's screen.

The most common use cases involve making fields / comments required on specific transitions. If a field is mandatory via the field configuration it's mandatory from the beginning. Usually a user can't possibly know all the issue details when creating it. Over the course of time some fields might become mandatory

If a validator fails, the issue does not progress to the destination status of the transition, and the transition's post functions are not executed. (We'll cover post functions in the next module.)

Common Validators

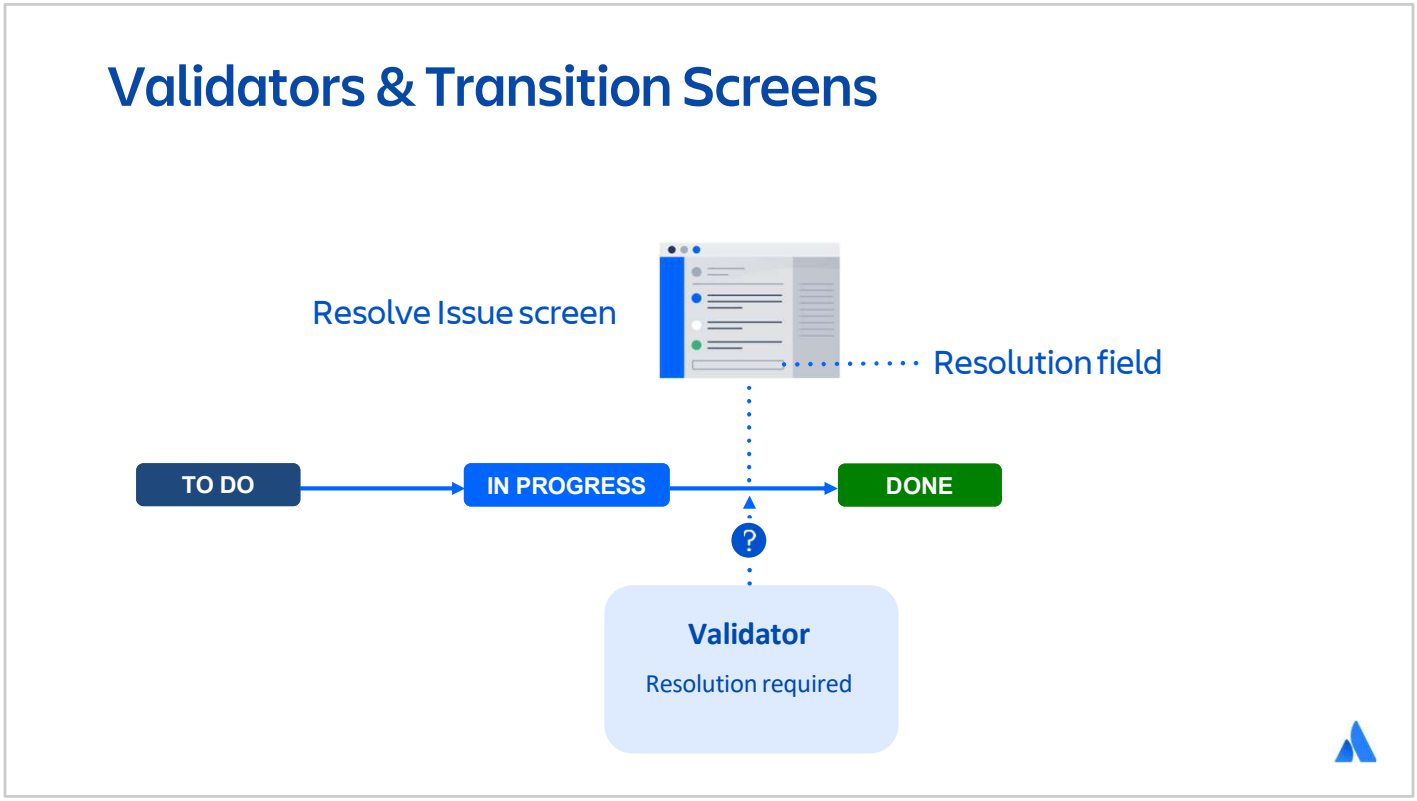
Validator	Example
Fields Required	Approver's name and comments when approve
Regular Expression Check	Email address contains @ symbol
Date Compare	Creation date < Due date

All of the following are common validators:

1. Fields Required - Used to ensure certain information is captured as part of a transition. An example of this is in an approval process where you need to capture the approver's name and comments when an issue is moved into the approved status.
2. Regular Expression Check - Useful when you need to validate that information has been entered in a certain format before an issue can be transitioned to the next status. You might use this when validating that an email address contains an @ sign.
3. Date Compare - Useful when you need to validate that an issue was created within a certain date window. You might use with an expenses process to validate that an issue's creation date is less than the Due Date for that month's expenses deadline.

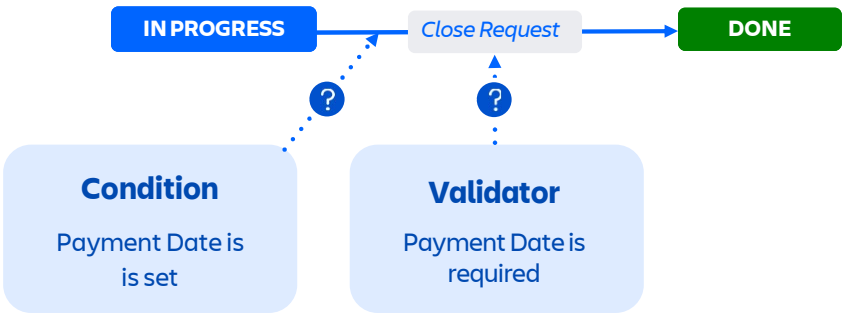
Note that these three validators are only available if you install the Suite Utilities for Jira app in Jira Server. (This was formerly called Jira Suite Utilities. And apps were formerly called add-ons on the Atlassian Marketplace.) Apps can give you additional conditions and flexibility with your workflows. New apps can be downloaded from the Atlassian Marketplace at marketplace.atlassian.com.

Or you can create your own conditions by building your own app. See https://developer.atlassian.com/display/JIRADEV/jira+developer+documentation and https://developer.atlassian.com/server/jira/platform/workflow-modules/ for details.



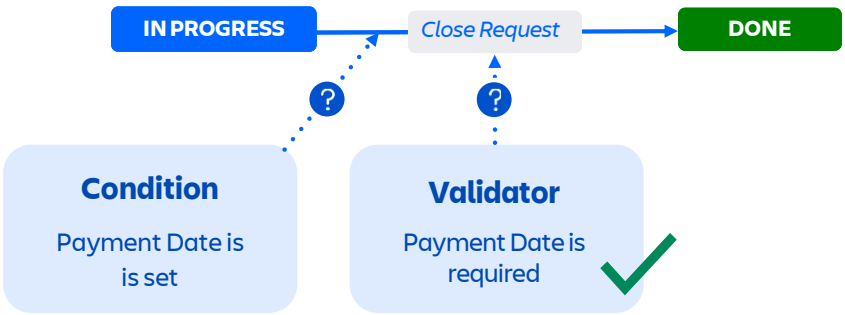
A transition screen can be used when information is required as an issue moves through a transition. When a user clicks on the transition button, a screen can be displayed in order to gather input from the user before the transition is executed. These pop-up screens can contain fields that don't appear on the default create or edit issue screens. For example, when an issue is resolved, users indicate how the issue was resolved via the Resolution field on the Resolve Issue screen. Transition screens provide a way to make fields mandatory at a point in your workflow that you select (in contrast to having them input when an issue is first created). This provides the flexibility to clear these field values on a certain transition and then make them mandatory again on a later transition within the workflow. To make a field on a transition screen mandatory, you can specify that in a validator that you add to the same transition as the screen.
Jira supplies transition screens that are built into the default Jira system workflow and the Resolve Issue screen is one of these. You can add these provided screens to your workflow transitions. You can also create your own custom screens and add them to your workflow transitions.

Condition or Validator?



Sometimes it can be difficult to decide whether to use a condition or a validator. For example, when closing a payment request, the payment date is mandatory. This could be a validator or a condition!
Answer is on the next slide.

Condition or Validator?



Answer: Using a validator is preferable as it reduces the effort to one process step. The user can close the issue and while they're doing this, add the payment date. (To do this the Payment Date field would need to be on a screen.) Otherwise the user would have to enter the date in the issue and then, in a second step, trigger the transition. Also users wouldn't even see that the Close Request transition exists until the payment date was entered. This could be confusing, especially for new hires.

CONDITIONS

Apply to a situation before a transition can take place

Can hide transitions from users

Questions to ask:

- Who should be able to see the transition?
- When should the transition be available?

VALIDATORS

Control if input is valid during a transition

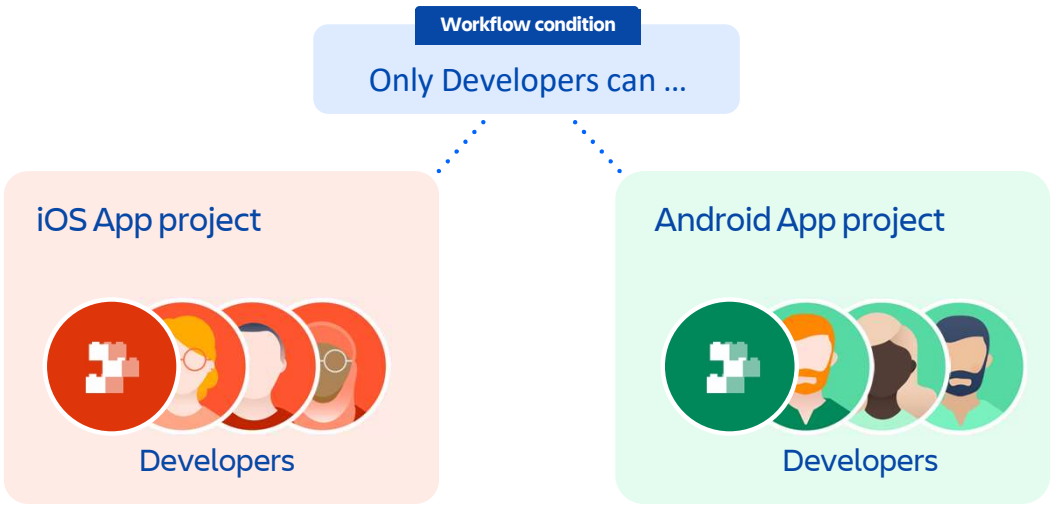
Can prevent the progression to target status

Questions to ask:

- What info has to be present or provided during the transition?
- Is the input valid?

Conditions control whether a transition is visible and can be executed, meaning that they are executed before a transition. Validators check the input during a transition, only if the conditions are met. An example of this is taking a certification course, where you must attend the course before you can take the exam. The condition is attending the course and the validation is that the score must be high enough in order to pass.
For more information, see https://confluence.atlassian.com/adminjiraserver/advanced-workflow-configuration-938847443.html (DC/Server) or https://support.atlassian.com/jira-cloud-administration/docs/configure-advanced-issue-workflows/ (Cloud).

Use Project Roles in Conditions & Validators



As a best practice, use project roles over groups, in your conditions and validators. This allows you to share workflows (and conditions/validators) across multiple projects and have different users meet or fail the permission problem depending on their membership in individual project roles. In this example two projects, iOS App and Android App, share the same workflow. In that workflow there are conditions and validators that use the project role Developers. Each project has different members in their Developers role. So as issues go through the workflow in each project the conditions and validators will be checked against different groups of users depending on what project the issue belongs to.

Groups are global across Jira so unless you created different groups for each project (which would not be a best practice) you couldn't achieve the same flexibility.

Conditions & Validators Best Practices

- Ensure your conditions or validators won't impact other workflows
- Only add necessary conditions and validators
- Keep conditions and validators simple
- Use out-of-the-box conditions and validators where possible
- Test, test, test!

Check whether a transition is shared before adding conditions or validators to it. For example, a shared approvals workflow that's used for approving holiday requests by the HR and product teams. If someone changes the transition from the Requested status to the Approve status so that only hr-managers can make this transition it breaks the workflow, as no one in the product-managers group is able to approve new feature requests.

Don't add too many conditions and validators. For example, when an approve transition is executed you have a Fields Required validator for every field. This is complex and hard to maintain when you could simply group all the required fields together in one Fields Required validator. Also, too many required fields and exceptions can get users frustrated.

Don't make your conditions and validators too complex else they'll be hard to maintain and manage. Some third party apps can be very powerful but can be complex to maintain requiring knowledge of, for example, scripting. Stick to the out-of-the-box conditions and validators when possible. In addition, conditions or validators can stop working if the app is removed, disabled or its license expires. However there are some very useful apps that will give you many new conditions, validators, post functions, etc.

Always test conditions and validators in a development instance before going live so you can ensure they work as expected. For example, adding a field required validator to the review code transition where you mandate that the UI Designer and Developer field are mandatory for this status. However, after the transition goes live you discover the UI Designer field is not available to your project meaning that no issues can be transitioned into your Review status.

See how it's done

- Restrict who can execute a transition with a condition
- Validate that a mandatory field contains data during a transition

Instructor demo.

Are you getting it?

To ensure that a Claims Specialist enters a valid reason when rejecting a claim, you would use a:

Condition

Validator

The answer is on the next slide.

Did you get it?

To ensure that a Claims Specialist enters a valid reason when rejecting a claim, you would use a:

Validator

Answer: To ensure an Assessor enters a valid reason when rejecting a claim, you'd use a Validator. Recall that validators validate input made during the transition.

Are you getting it?

To ensure only users in the Managers project role can submit a pay change request, you would use a:

Condition

Validator

The answer is on the next slide.

Did you get it?

To ensure only users in the Managers project role can submit a pay change request, you would use a condition.

Condition

Answer: To ensure only users in the Managers project role can submit a pay change, request you'd use a condition in the transition for the pay change request. Recall that a condition must exist before the transition is permitted.

Are you getting it?

As a best practice, use _____ in your conditions and validators.

a. Individual users
b. Groups
c. Project roles

The answer is on the next slide.

Did you get it?

As a best practice, use _____ in your conditions and validators.

a. Individual users
b. Groups
✓ c. Project roles

Answer: c Project roles

Are you getting it?

Validators can:

a. Hide transitions from users
b. Prevent the progression of an issue to the target status
c. Be grouped and nested

Validators can:

a. Hide transitions from users

b. Prevent the progression of an issue to the target status

c. Be grouped and nested

Did you get it?

a. Hide transitions from users
✓ b. Prevent the progression of an issue to the target status
c. Be grouped and nested

Answer: b Validators can prevent the progression of an issue to the target status.

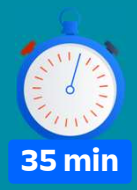
Takeaways

- Use project roles instead of groups in conditions and validators
- Only add necessary conditions and validators and keep them as simple as possible

As a best practice, use project roles over groups, in your conditions and validators. This allows you to share workflows (and conditions/validators) across multiple projects and have different users meet or fail the permission problem depending on their membership in individual project roles.
Don't add too many conditions and validators. You want to mandate certain fields when an approve transition is executed by having a Fields Required validator for every field. This is complex and hard to maintain when you could simply group all the required fields together inside one Fields Required validator. Additionally, too many conditions and validators have a negative impact on the user experience. If a workflow is too complex (many required fields and exceptions) users to get frustrated. Don't make your conditions and validators too complex else they'll be hard to maintain and manage. Some third party apps can be very powerful but can be complex to maintain requiring knowledge of, for example, scripting or programming.

Lab 3 – Creating Conditions & Validators

- Exercise 1:
  - Add a complex condition using grouping
- Exercise 2:
  - Create a transition screen
  - Add a validator that requires a comment on the transition screen
- Optional Exercise 3:
  - Resolve a workflow problem



4 Automating Your Workflow with Post Functions

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

Triggering Transitions

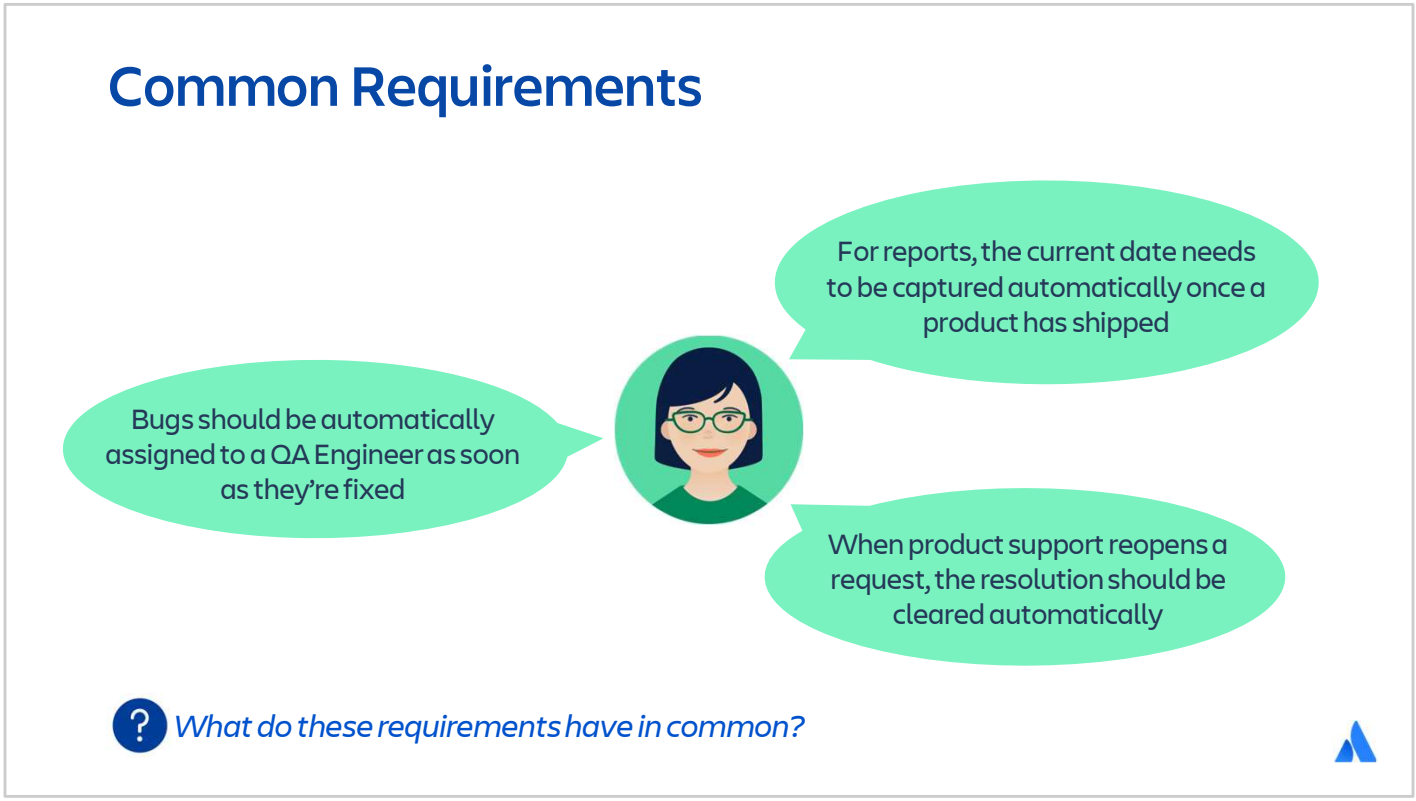
Extending Workflows with Properties

Taking It to the Next Level

What will you learn?

- Automate your process tasks by creating post functions

Here you'll learn how to automate certain processes by creating post functions. This makes the workflow process more efficient for your users.

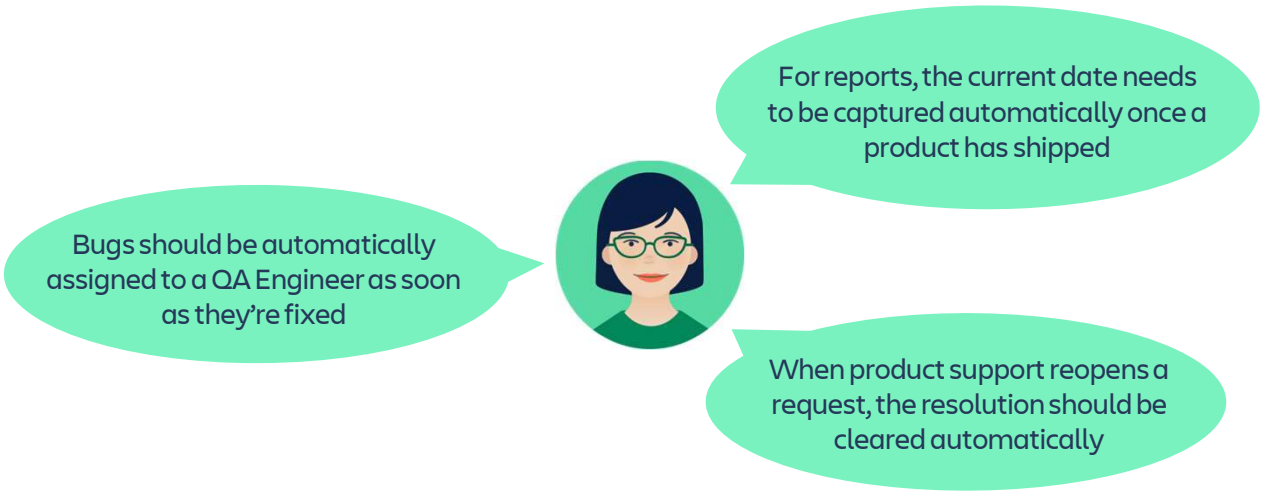


Common Requirements:

1. Bugs should be automatically assigned to a QA Engineer as soon as they're fixed.
2. For reports, the current date needs to be captured automatically once a product has shipped.
3. When product support reopens a request, the resolution should be cleared automatically.

What do these requirements have in common?

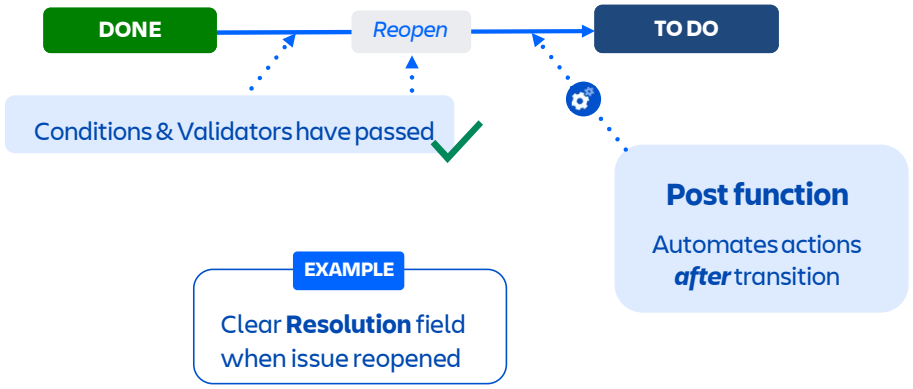
Common Requirements



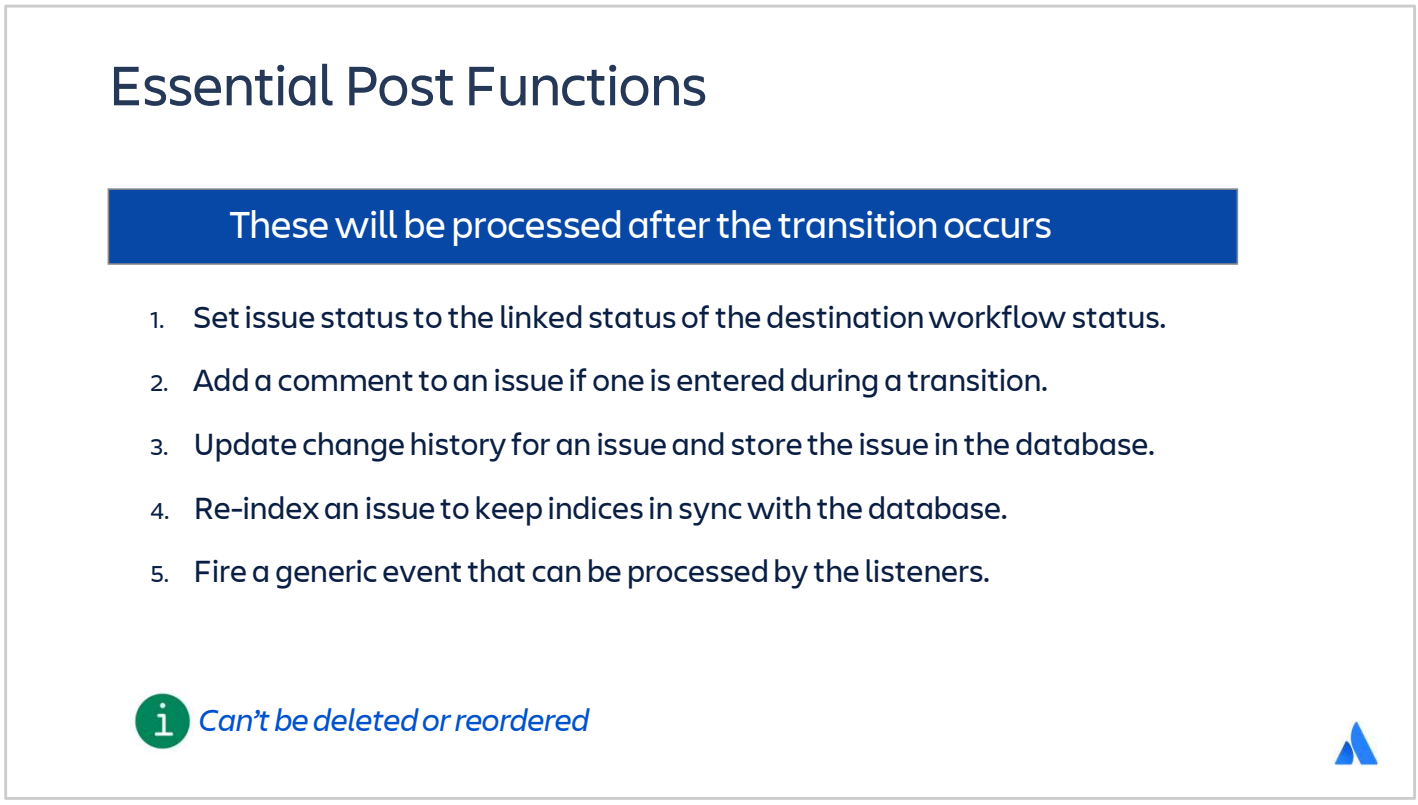
These are automated actions performed after something has occurred

Answer: These are all automated actions that need to take place once a transition has executed. This can be achieved by using post functions.

Post Functions



A post function is an automated action after a transition has executed.
Post functions carry out any additional processing required after a transition is executed.
A pre-requisite is that the conditions and validators have been passed.
The example shown here is to clear the Resolution field after the Reopen transition has executed. The issue is being transitioned from DONE back to TO DO.

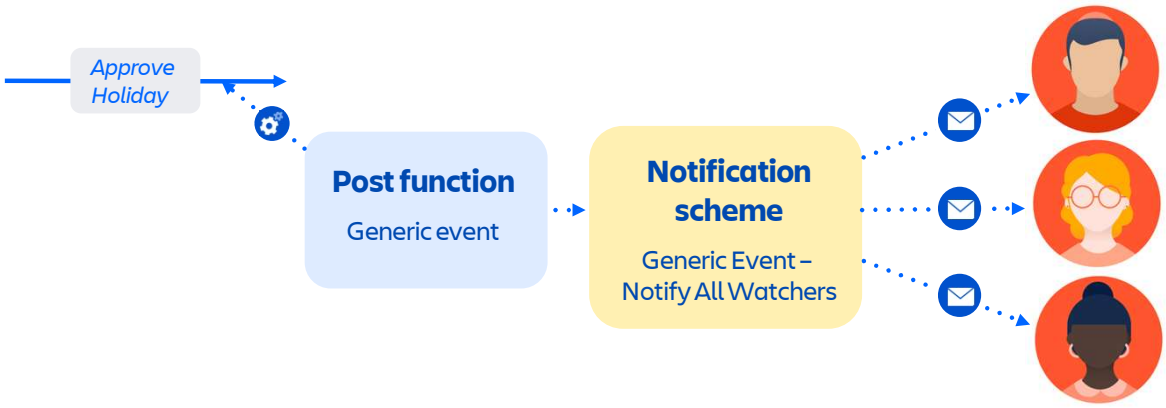


Every Jira transition has the following essential post functions, which are performed in this order:

1. Set issue status to the linked status of the destination workflow status.
2. Add a comment to an issue if one is entered during a transition.
3. Update change history for an issue and store the issue in the database.
4. Re-index an issue to keep indices in sync with the database.
5. Fire a generic event that can be processed by the listeners. Events in Jira allow you to trigger an action, such as sending out a custom notification after a successful transition. See the next slide for an example.

These essential post functions cannot be deleted from a transition or reordered. However, you can insert other (optional) post functions between them.

Using an Event to Send Email Notifications



Jira uses an event-listener mechanism to alert the system that something has happened, and to perform appropriate action (e.g. send an email notification) based on that event.
You can use a generic event in a post function to send email notifications. Use the 'Fire a generic event that can be processed by the listeners' essential post function to fire the 'Generic Event'. The generic event is a built-in Jira event can trigger the sending of email notifications after a particular transition is executed. In this example the Approve Holiday transition has a post function that fires a generic event. The notification scheme has a notification set for All Watchers for the Generic Event. So the three watchers will be notified when the holiday has been approved.
This is an example of a custom event. There are also system events which are used throughout Jira internally and cannot be added or deleted.
When a transition is performed, Jira will:

1. Look up the notification scheme associated with the issue's project and identify the users associated with the fired event
2. Send an email notification to each user

The fired event is also propagated to all registered listeners.
Notification schemes map events and email recipients.
For more information, see https://confluence.atlassian.com/adminjiraserver/advanced-workflow-configuration-938847443.html (Server/DC) or https://support.atlassian.com/jira-cloud-administration/docs/configure-advanced-issue-workflows/ (Cloud).

Common Optional Post Functions

Assign issues

Update fields

Resolution: **Fixed**

Ensure resolution is set by either a post function or in a transition screen

Jira includes several optional post functions that can be added to transitions, for example:

- Assigning issues (to current user, lead developer, reporter). For example, assigning an issue back to the reporter when their holiday request has been rejected.
- Updating an issue's field. For example, setting the resolution - either to clear it when transitioning an issue back to an unresolved state or to set it when transitioning an issue to DONE. See more information below. The fields that can be updated include Assignee, Description, Environment, Priority, Resolution, Summary, Original Estimate, and Remaining Estimate. This post function cannot update custom fields.

Other optional post functions include sending notifications to one or more Hipchat rooms, create Perforce Job Function, etc. For a full list of the optional post functions, see https://confluence.atlassian.com/adminjiraserver/advanced-workflow-configuration-938847443.html (DC/Server) or https://support.atlassian.com/jira-cloud-administration/docs/configure-advanced-issue-workflows/ (Cloud). Jira sees the Resolution field as a flag. If it has a value, then the issue is resolved (closed). If it's empty, then the issue needs some form of action (open). This is very important as Jira uses this when running reports. So for any transition going to a status that you think of as "closed", make sure that you either use a workflow post function or put the resolution on the workflow transition screen so the user has to fill it in. Likewise, the resolution needs to be removed if an issue flows back into an unresolved state.

Ordering Post Functions

1. **The Resolution of the issue is cleared.**
2. Set issue status to the linked status of the destination workflow status.
3. **Assign the issue to the reporter.**
4. Add a comment to an issue if one is entered during a transition.
5. ...

⚠ *Be careful with the sequence!*

Optional post functions can be added and inserted between essential post functions. You can use optional post functions to meet specific requirements. For example, here we see two optional post functions that are inserted in front of and between the essential post functions.
We said earlier that the sequence of essential post functions can't be changed. That holds true, BUT: you can add as many optional post functions between them as you want.
But be careful with the sequence. See example on the next slide.

Ordering Post Functions Example

...

4. Re-index an issue to keep indices in sync with the database.
5. Fire a generic event that can be processed by the listeners.
6. Assign the issue to the reporter.
7. The field **Approver** will take the value from **Assignee**.

What's wrong with this sequence?

There are two problems with the sequence of post functions in this example. Can you see what's wrong? The answer is on the next slide.

Ordering Post Functions Solution

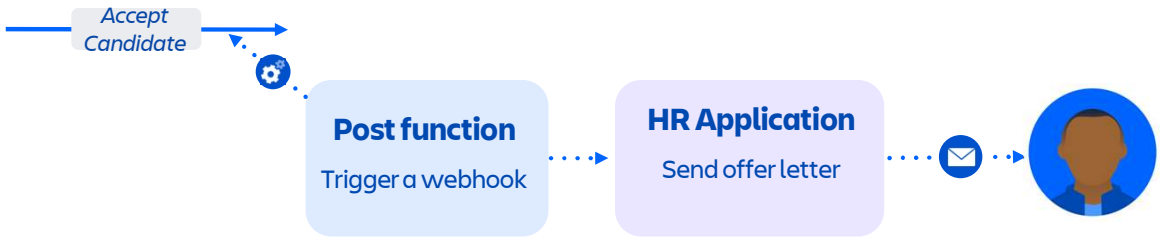
...

4. The field **Approver** will take the value from **Assignee**.
5. Assign the issue to the reporter.
6. Re-index an issue to keep indices in sync with the database.
7. Fire a generic event that can be processed by the listeners.

There are two problems with this order which we've fixed:

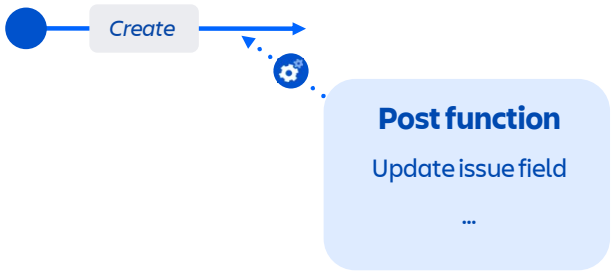
1. Custom post functions (blue) have been reversed. You want the value of the Assignee to be copied to the Approver field before you assign the issue to the reporter for a final check.
2. The first five post functions are the essential post functions and you can't reorder those. But the custom functions have been moved above the Re-index essential post function (black). You should always update any field values before indexing issues. Indexing, database update and events should be the final post functions.

Adding Webhooks to Post Functions



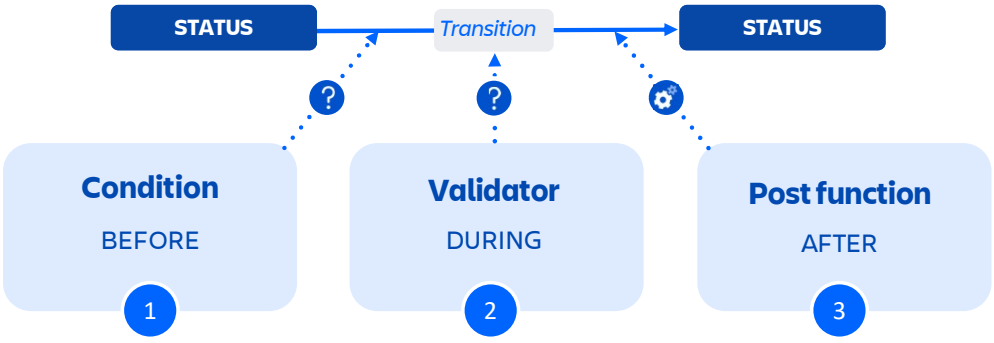
If you have webhook defined in Jira, you can add one of the optional post functions (Trigger a Webhook) to trigger the webhook during the workflow transition. A webhook is a user-defined callback over HTTP. You can use Jira webhooks to notify your app or web application when certain events occur in Jira. For example, you want to trigger an action in your HR application (Create offer letter) when an issue representing a potential new hire goes through the Accept candidate transition.
Triggers the specified webhook after completing the workflow transition. When you add this post function, you will be asked to specify a webhook. This webhook must already be defined in Jira. See https://confluence.atlassian.com/adminjiraserver/managing-webhooks-938846912.html (Server/DC) or https://confluence.atlassian.com/display/AdminJIRACloud/Managing+webhooks (Cloud) and https://developer.atlassian.com/jiradev/jira-apis/webhooks.

Adding Post Functions to Initial Transition



You can add post functions to the initial Create transition. This can be useful if you need to perform processing tasks when an issue is created. Such as setting a particular field's value. For more information, see https://confluence.atlassian.com/adminjiraserver/advanced-workflow-configuration-938847443.html (Server/DC) or http://go.atlassian.com/cloudadvwfs (Cloud).

Conditions vs. Validators vs. Post Functions



Conditions, validators and post functions are processed in order

Conditions and validators analyze (status / input) and can prevent the execution of a transition

Post functions automate actions

Post Functions Best Practices

- Ensure post functions are in the correct order
- Keep post functions to a minimum, simple and maintainable
- Ensure your post functions won't impact other workflows
- Use out-of-the-box post functions where possible
- Use post functions to set and clear resolutions
- Test, test, test

Ensure post functions are in the correct order e.g. don't index an issue before updating a field.
Ask yourself if a Post Function is absolutely necessary before adding it. For example, adding several post functions to automatically set field values on a workflow transition when these fields may not be relevant to the business process that the workflow is modeling. Keep post functions simple and avoid using complex post functions which will be hard to maintain.
Be careful when adding post functions to shared transitions or shared (active) workflows as there might be unwanted effects. For example, two projects using a dev workflow. One project might want notifications being sent to a Hipchat room, the other doesn't.
Use out of the box post functions where possible to avoid reliance on third party apps. Dependence on third party apps can cause post functions to stop working if the app is removed, disabled or its license expires. Don't delete a workflow enhancing app without analyzing their usage.
Set a resolution automatically to facilitate standardized reports. Also clear resolutions automatically when an issue is reopened. This keeps your data accurate.
Test post functions in a development instance first before going live so you can ensure they work as expected. We recommend you keep a clone of your project and workflow in a development environment, where you can verify that new post functions in will not break the workflow before adding them to production.

See how it's done

- Add a post function to a transition

Instructor demo

Are you getting it?

When an issue is being approved, to update the Approval Date field with the current date you would use a:

Validator

Post function

When an issue is being approved, to update the Approval Date field with the current date you would use a validator or a post-function?

Did you get it?

When an issue is being approved, to update the Approval Date field with the current date you would use a post function.

Post function

Answer: To update the Approval Date field with the current date you would use a post function. This is something that can be handled automatically after the transition has occurred.

Are you getting it?

When an issue is being approved, to ensure the Finance Manager field is filled in you would use a:

Validator

Post function

When an issue is being approved, to update the Approval Date field with the current date you would use a validator or a post-function?

Did you get it?

When an issue is being approved, to ensure the Finance Manager field is filled in you would use a validator.

Validator

Answer: To ensure the Finance Manager field is filled in while moving it to the approved state you would use a validator. This is something that occurs during the transition.

Are you getting it?

The resolution can be set using:

Transition screen

Post function

How can the resolution be set? Using a transition screen or a post-function?

Did you get it?

The resolution can be set using:

Transition screen

Post function

Answer: The resolution can be set using either a transition screen (where the user selects the appropriate resolution) or automatically by a post function.

Takeaways

- Keep post functions to a minimum, simple and maintainable
- Use post functions to set and clear resolutions
- Test, test, test

Ask yourself if a Post Function is absolutely necessary before adding it. For example, adding several post functions to automatically set field values on a workflow transition when these fields may not be relevant to the business process that the workflow is modeling. Keep post functions simple and avoid using complex post functions which will be hard to maintain.
Set a resolution automatically to facilitate standardized reports. Also, clear resolutions automatically when an issue is reopened. This keeps your data accurate.
Test post functions in a development instance first before going live so you can ensure they work as expected. We recommend you keep a clone of your project and workflow in a development environment, where you can verify that new post functions in will not break the workflow before adding them to production.

Lab 4 – Automating Your Workflow with Post Functions

- Exercise 1
  - Add post functions to set and clear resolution
- Exercise 2
  - Add post functions to copy field values
- Optional Exercise 3
  - Read up on post functions

5 Triggering Transitions

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

Triggering Transitions

Extending Workflows with Properties

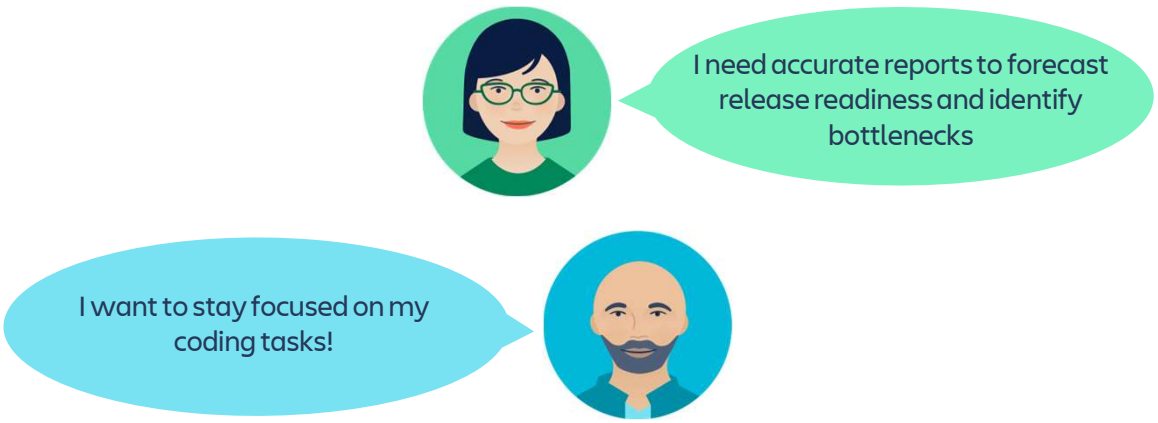
Taking It to the Next Level

What will you learn?

- Automate transitions by creating triggers

Here you'll learn how to automate your workflow transitions by creating triggers from connected Dev tools. This allows your developers to focus on their coding tasks.

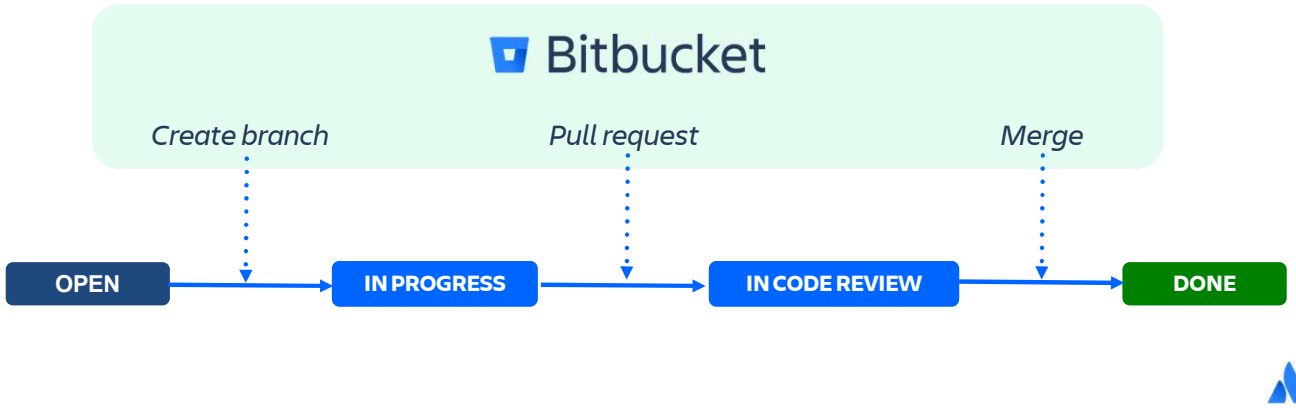
Software Teams



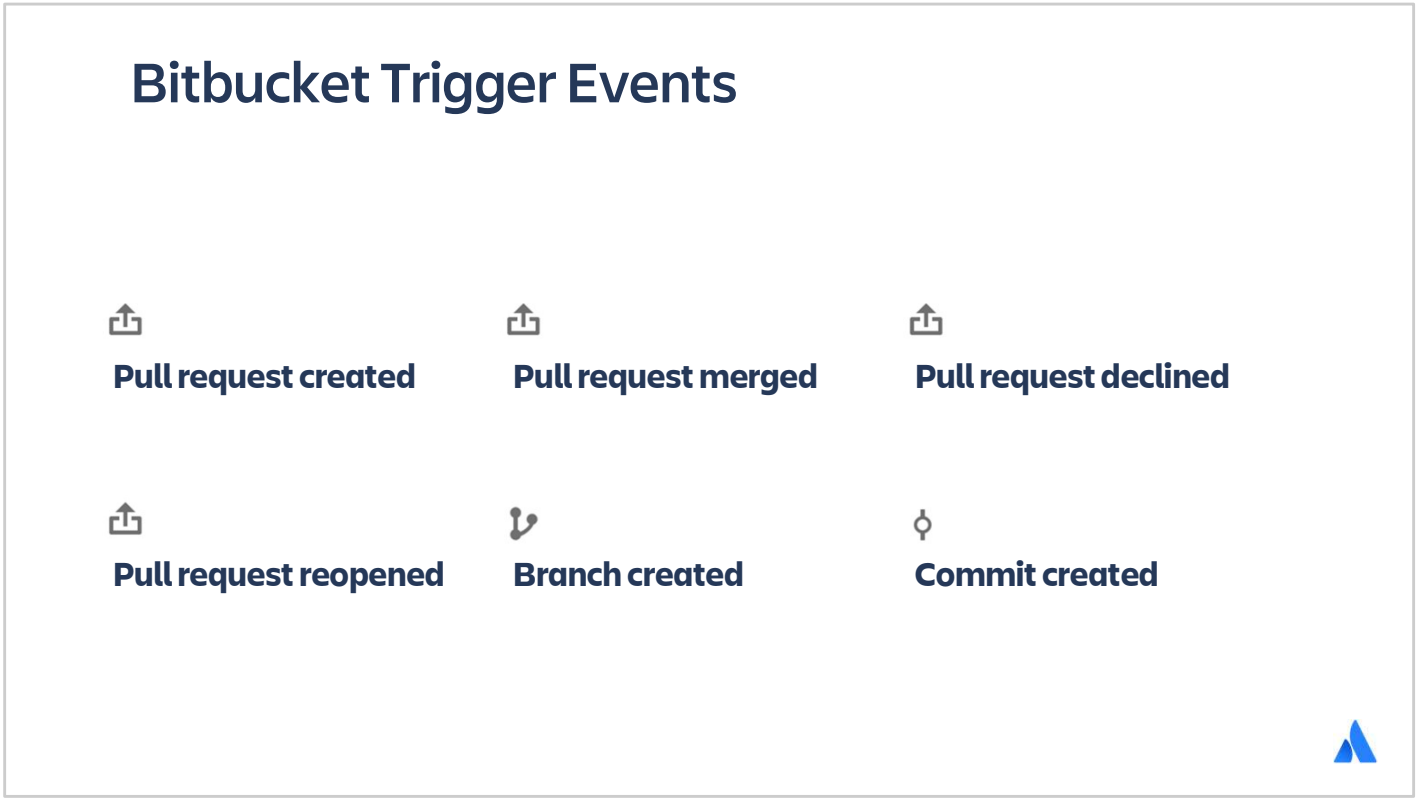
Software teams need to release product as fast as possible. Developers often work in multiple development tools including Bitbucket, Crucible, etc. They need to keep focused on their development tasks and not have to worry about updating their issues. And project leads need to know how development tasks are progressing and where any bottlenecks might be so they can address them quickly. They also need to be able to rely on their reports such as their sprint burndown charts to ensure the project is on track and they can forecast when they'll be ready to release their product.
How can they meet the needs of the team and keep their issue status up to date?

Triggers

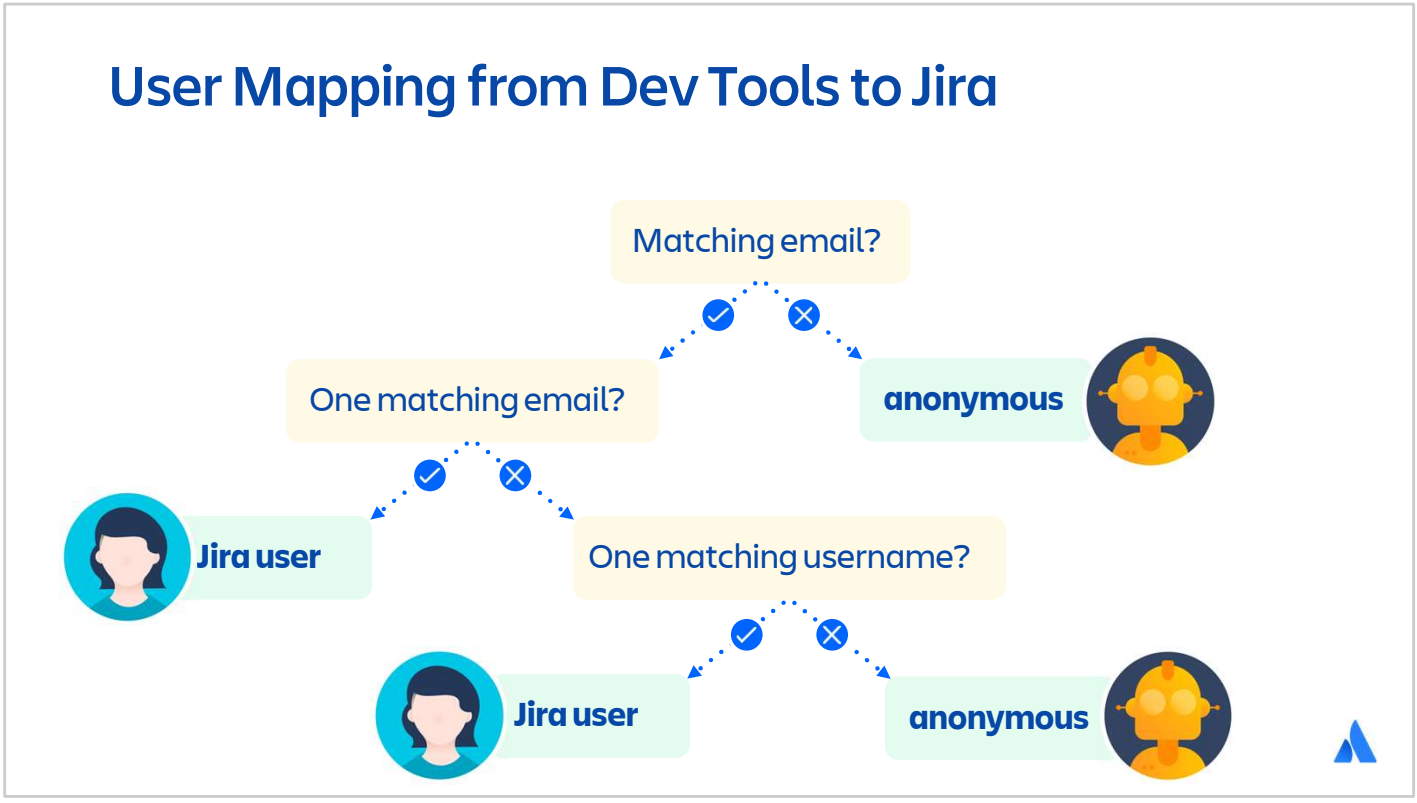
Automatically transition issues when events occur in your dev tools



Jira admins can configure triggers in workflows to automatically transition issues when events such as creating branches, pull requests and merges occur in development tools. This saves developers time and let's them concentrate on their coding tasks.
Jira admins can configure triggers that respond to events in your linked development tools such as Bitbucket, Fisheye, Crucible, and Github. For example, when a developer creates a branch to start work on an issue in Bitbucket, the issue will automatically be transitioned from 'Open' to 'In progress'. Then when the developer creates a pull request for other developers to do code review, the issue will automatically be transitioned to 'In Code Review'. And finally when the issue is merged in Bitbucket the issue is considered Done.
So issue statuses are updated automatically based on activity in your code base. This means developers can remain focused on their coding task, knowing that Jira is automatically keeping their teammates and stakeholders up to date. And because those updates happen in real time, project leaders get hyper-accurate burn-down charts to help them forecast release readiness and identify bottlenecks in their team's processes.
You can still drag issues between columns on the agile board or update your status using the workflow buttons in the issue. Workflow automation triggers simply provide more options for how issues move from one status to the next, as well as provide assurance that the current status reflects the actual state of development.
Before using triggers, your Jira admin needs to connect your development tools to Jira.



These are the Bitbucket events that can trigger an automatic workflow transition in Jira. There are other events available for other development tools. For a full list, see https://confluence.atlassian.com/adminjiraserver/configuring-workflow-triggers-938847513.html (Server/DC) or http://go.atlassian.com/cloud_wftriggers (Cloud). Also see this document for how to reference a Jira issue in various events.

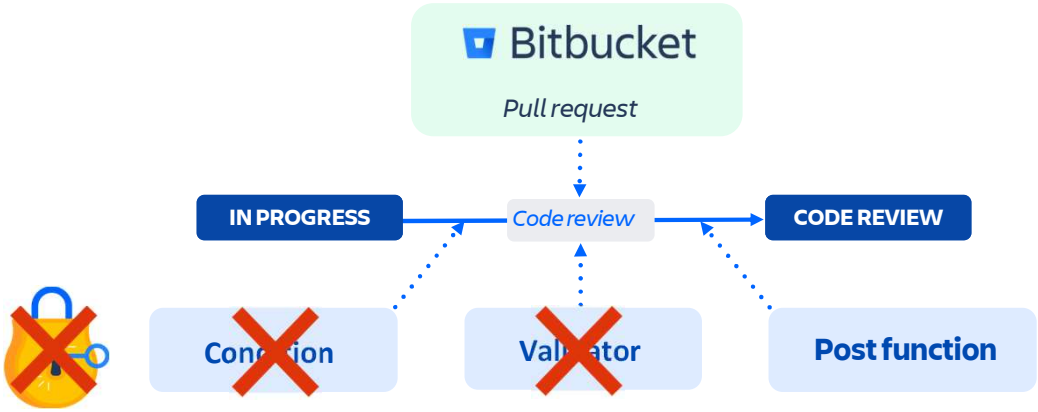


The development tool user is mapped to a Jira user for workflow triggers by matching the email address, then the username. The process is:

- Single Jira user with a matching email address – Transition the issue as the Jira user.
- No Jira users with a matching email address – Transition the issue as an anonymous user.
- Multiple users with a matching email address in Jira – Try to find a matching username in that group of users. If there is a Jira user with a matching username, transition the issue as the Jira user. If there is no matching username, transition the issue as an anonymous user.

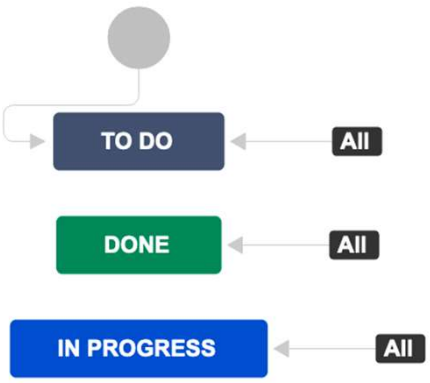
For details on each development tool, see https://confluence.atlassian.com/adminjiraserver/configuring-workflow-triggers-938847513.html (Server/DC) or http://go.atlassian.com/cloud_wftriggers (Cloud).

Things to Watch Out For



- *Permissions, conditions, and validators are ignored for automatic transitions*
- *Careful of post functions requiring users executing as anonymous*

When a transition is triggered automatically, it ignores any conditions, validators or permissions configured on the transition.
When workflow transitions are automated with triggers, permissions around who can move the issue between states are ignored. Same with conditions and validators. But conditions, validators, and permissions will still apply for that transition when it's executed manually.
However, post functions are still executed. You need to be careful that if your post function requires a user, that your transition will not be executed by an anonymous user (see previous slide on user mapping).



Avoid using triggers on global transitions

Use caution when combining triggers and global transitions. For example, using the commit trigger with a global transition that moves issues (from whatever state they're currently in) to In Progress is fraught with peril. Commits are often made during the code review phase in order to incorporate the feedback from your reviewers–in which case, you'd want the issue to remain in an In Review state.

We recommend that you do not configure triggers for global transitions, unless you are confident that you understand exactly how the trigger will affect the behavior of the issue.

Configuring triggers for global transitions can often result in an issue unexpectedly transitioning to the target status for the global transition. For example, consider if you configured a 'Commit created' trigger for the global transition to the 'In Progress' status. Committing code can happen at many stages during an issue's lifecycle (e.g. writing the initial code, changing code after a review, etc.) This could result in the issue incorrectly transitioning to 'In Progress' out of a number of statuses, like 'In Review' or 'Done'.

Tip: If you do use global transitions in your workflow, you will probably have multiple transitions into a status. This means that users will have multiple workflow options on an issue (e.g. both 'Start Progress' and 'In Progress'). To hide options, add the 'Hide transition from user' condition to the relevant transitions.

Recall that a global transition allows any status in a workflow to transition to any other status in the workflow. You'll see the All indicator.

Are you getting it?

Which of these are ignored for automatic transitions?

a. Conditions
b. Validators
c. Post functions
d. Permissions

Which of these are ignored for automatic transitions?

a. Conditions

b. Validators

c. Post functions

d. Permissions

Did you get it?

Which of these are ignored for automatic transitions?

- ✓ a. Conditions
- ✓ b. Validators
- c. Post functions
- ✓ d. Permissions

Answer: a, b, and d
Conditions, validators, and permissions are ignored when an automatic transition occurs as part of a trigger.

Are you getting it?

A trigger is configured for the **Start progress** transition to automatically execute when a branch is created in Bitbucket. Which of these can cause the Start progress transition to execute (updating the issue status to In Progress)?

a. Dragging the issue on the board to the In Progress column
b. Clicking the Start progress transition button in the issue
c. Creating a branch in Bitbucket, that includes the issue key
d. Creating a branch from the issue in Jira

A trigger is configured for the **Start progress** transition to automatically execute when a branch is created in Bitbucket. Which of these can cause the Start progress transition to execute (updating the issue status to In Progress)?

a. Dragging the issue on the board to the In Progress column

b. Clicking the Start progress transition button in the issue

c. Creating a branch in Bitbucket, that includes the issue key

d. Creating a branch from the issue in Jira

Did you get it?

A trigger is configured for the **Start progress** transition to automatically execute when a branch is created in Bitbucket. Which of these can cause the Start progress transition to execute (updating the issue status to In Progress)?

- ✓ a. Dragging the issue on the board to the In Progress column
- ✓ b. Clicking the Start progress transition button in the issue
- ✓ c. Creating a branch in Bitbucket, that includes the issue key
- ✓ d. Creating a branch from the issue in Jira

Answer: a, b, c, and d
All these actions will cause the Start progress transition to execute and update the issue's status to In Progress.

Are you getting it?

To ensure a notification is sent to the reporter of the bug when the bug is closed, you would use a:

Trigger

Post function

To ensure a notification is sent to the reporter of the bug when the bug is closed, you would use a trigger or a post-function?

Did you get it?

To ensure a notification is sent to the reporter of the bug when the bug is closed, you would use a post function.

Post function

Answer: To ensure a notification is sent to the reporter of a bug when the bug is closed, you'd use a post function. Recall that post functions are automated actions that occur after the transition has occurred. Whereas triggers automatically transition Jira issues when certain events occur in a connected developer tool such as Bitbucket.

Takeaways

- If you're using post functions that require user information, ensure your users won't be mapped as anonymous
- Don't configure triggers for global transitions

You need to be careful that if your post function requires a user, that your transition will not be executed by an anonymous user (see slide on user mapping).
We recommend that you do not configure triggers for global transitions, unless you are confident that you understand exactly how the trigger will affect the behavior of the issue.

There is no lab on this module.

6 Extending Workflows with Properties

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

Triggering Transitions

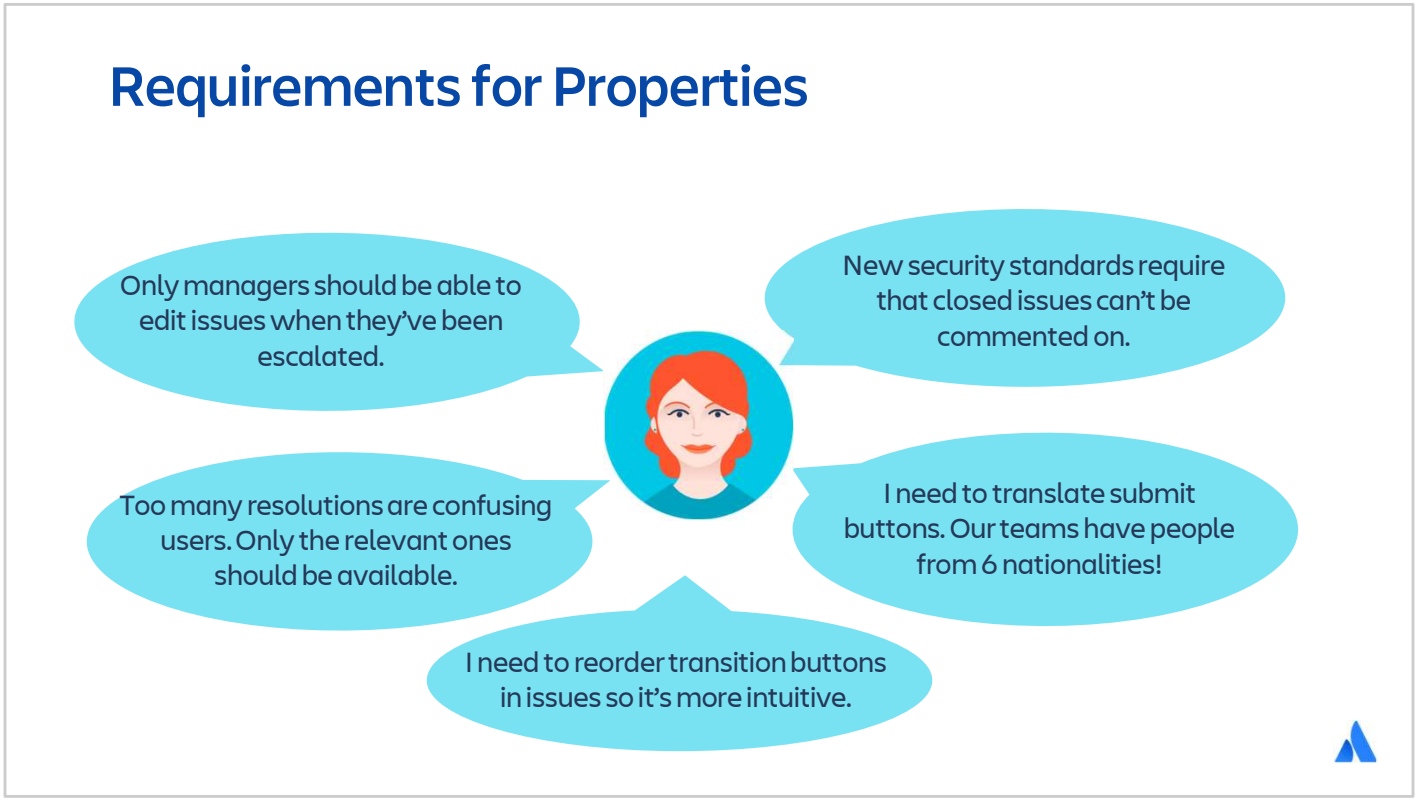
Extending Workflows with Properties

Taking It to the Next Level

What will you learn?

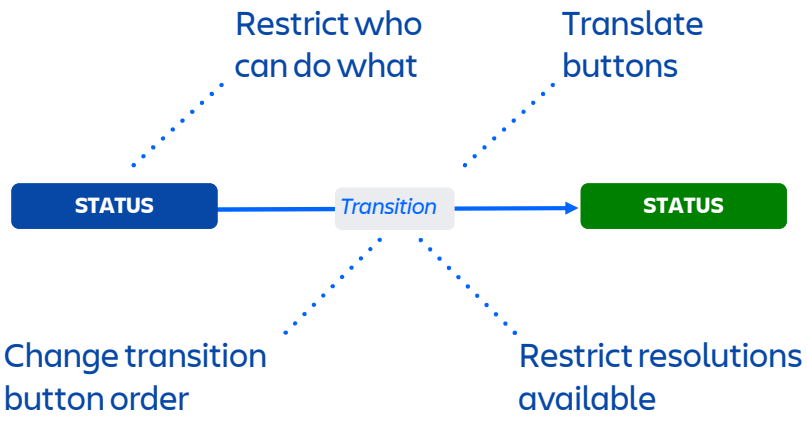
- Restrict who can do what in certain statuses
- Reorder transition buttons
- Control which resolutions are available

Here you'll learn how further customize and enrich your workflows by creating properties. You'll learn how to create properties to restrict who can do what in certain statuses, reorder transition buttons on the view issue screen, and control which resolutions are available to users.

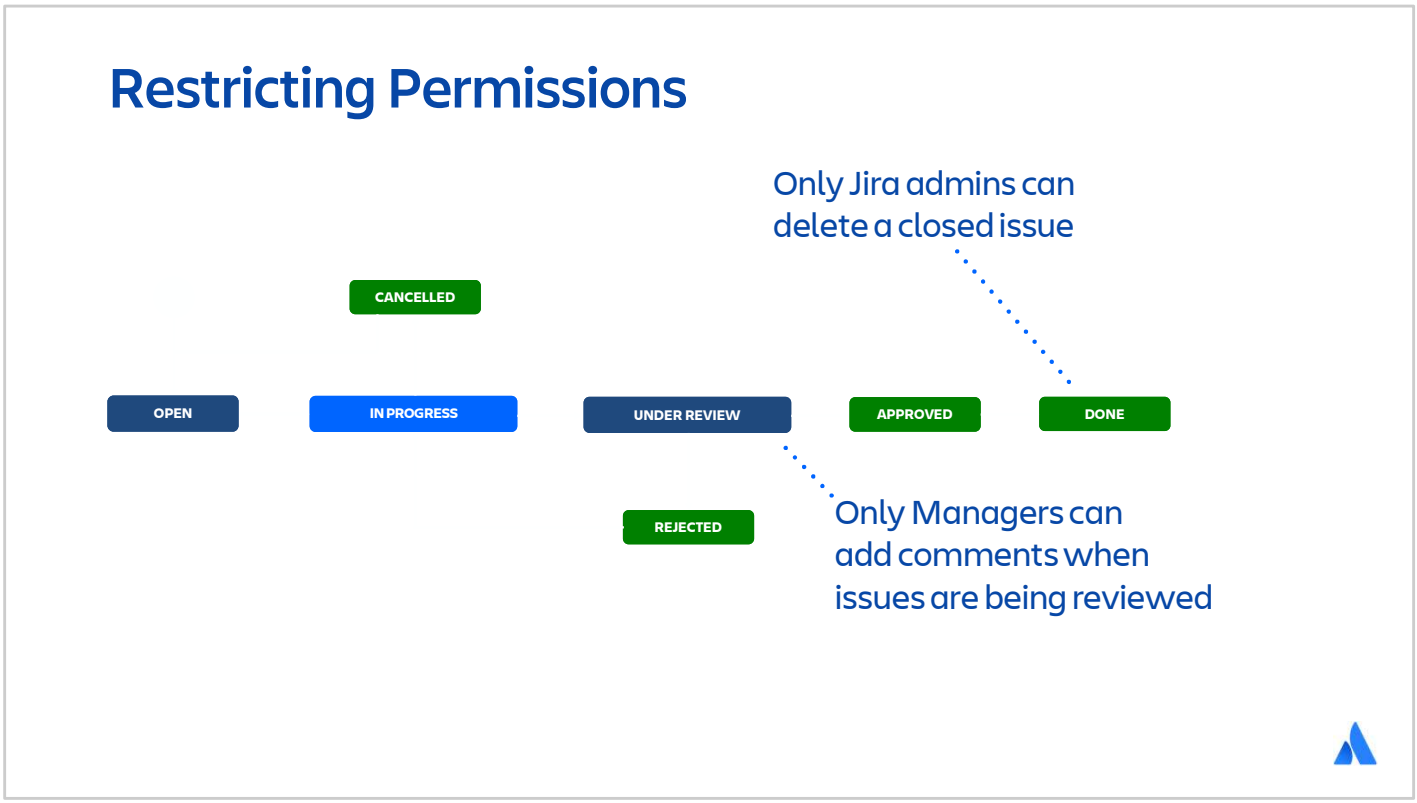


These are all requirements for properties. Most of the properties are quite different in what they can do.

Enriching Your Workflow with Properties



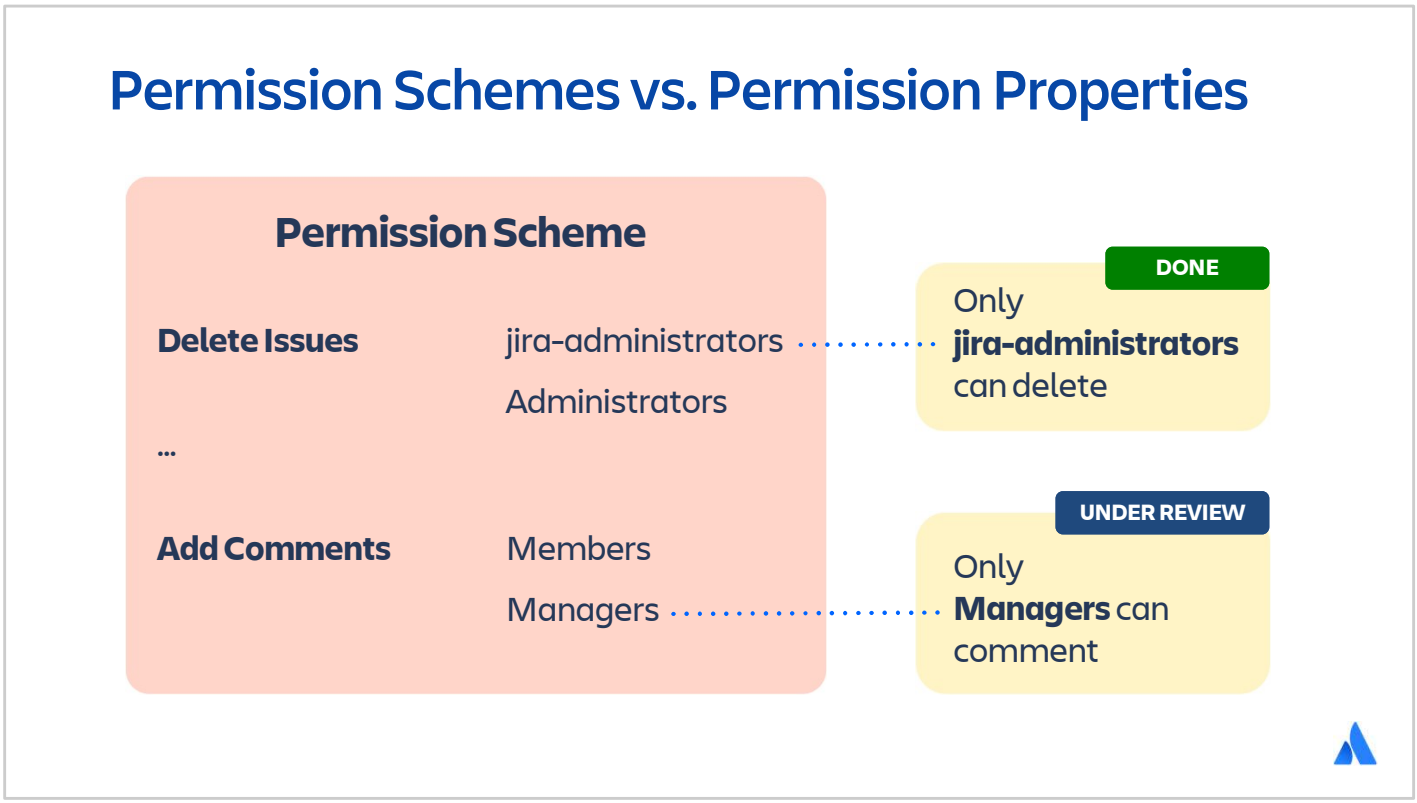
Properties are a hidden gem of Jira. You can use them to further customize and enrich your workflow. For example, you can implement restrictions on who can do what in a certain status. Or restrict what resolutions are available in a transition. Or even reorder the transition buttons within issues or translate buttons. But be careful when using these as maintaining them and sharing workflows that use them can be tricky. Let's look at each property you can use in the next slides.



Using the 'permission' workflow property, you can restrict permissions at the status level. You can restrict various permissions to a project role, group, user, and more when an issue is in that status. You restrict WHO can do what to an issue and WHEN they can do it.

For example, in this business workflow, only users in the Managers project role can comment on issues when they're being reviewed. Also only users in the jira-administrators group can delete closed issues.

How to set this property will be covered later in this module.



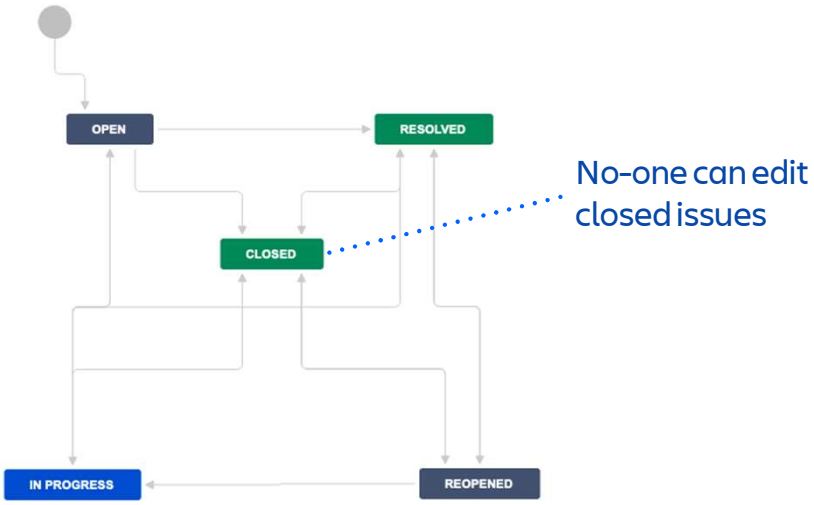
Permission schemes are used to set many different kinds of permissions that control what users can do in projects and with issues, comments, attachments, etc.
Workflow properties allow you to more narrowly restrict permissions. For example, in the permission scheme shown here, only Jira admins and project admins can delete issues but using a property in the workflow, you restrict delete permission to just Jira admins, and further, only when issues are closed (in the DONE status). The permission scheme controls deleting an issue at any point in the workflow. Whereas the property controls deleting an issue at a particular point in the workflow. And further, the property restricts who can do it in that status. So it gives you finer control. Also in the permission scheme shown here, only users in the Members or Managers project roles can add comments to issues. But the property narrows that permission so only users in the Managers project role can comment on issues in the UNDER REVIEW status.
Note that enforcing permissions at the status level can only restrict permissions set in the permission scheme, not grant permissions. In this example, you couldn't grant the Members project role permission to edit closed issues by adding them to the property.
In most cases, you should be handling permissions through the permission scheme.

Permission Property Best Practice

- Use project roles in permission properties to make maintenance easier

It's a good idea to use project roles in the permission property whenever possible. That way the users who are affected by this property will be updated automatically as the members of project roles change. If you use individual users, you would manually need to update the properties each time a change was needed, for example, a user left the company. Also project roles are preferable to groups as workflows may be shared between multiple projects and so the members of the roles will be different in each project.

Making Issues Read-Only

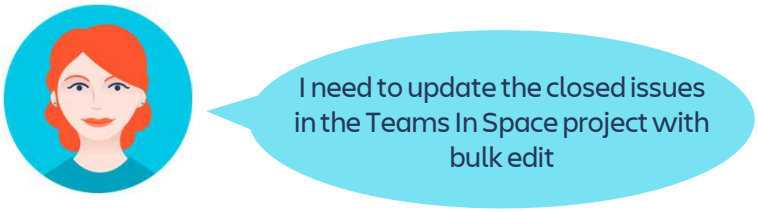


You can use the ‘editable’ property to prevent issues from being edited when they are in a particular workflow status. Here you see the Jira system workflow. By default this has the editable property set on the CLOSED status so that once an issue is closed it cannot be edited. (This is not set on the default simplified workflows that you get with many projects.) This is different from using the permission property we just discussed which restricts editing of an issue to a project role, group, etc. With this editable property, no-one can edit the issue.

Note that if issues are not editable in the project’s permission scheme, you cannot make them editable by setting the editable property to true. You can only restrict permissions.

How to set this property will be covered later in this module.

Making Issues Read-Only



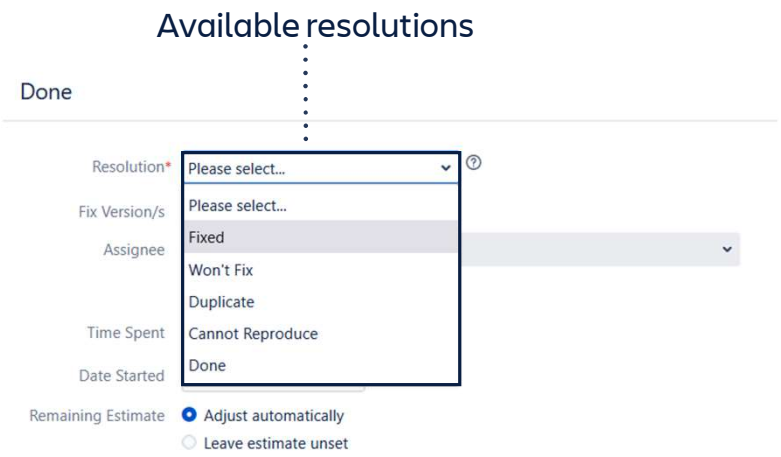
What would happen if issues in the Closed status were read-only?

Issues which cannot be edited cannot be updated using Bulk Edit either. In this situation the Jira admin wants to perform a bulk edit on the closed issues in the Teams In Space project. If the editable property has been used to made issues in the Closed status read-only then she will not be able to perform a bulk edit operation on them.
Instead, of making issues read-only, make them only writable to Jira admins, that way they can still perform bulk updates if needed.

Controlling Which Resolutions are Available

Resolutions defined

Fixed
Won't Fix
Duplicate
Incomplete
Cannot Reproduce
Unresolved
Done
Won't Do
Declined



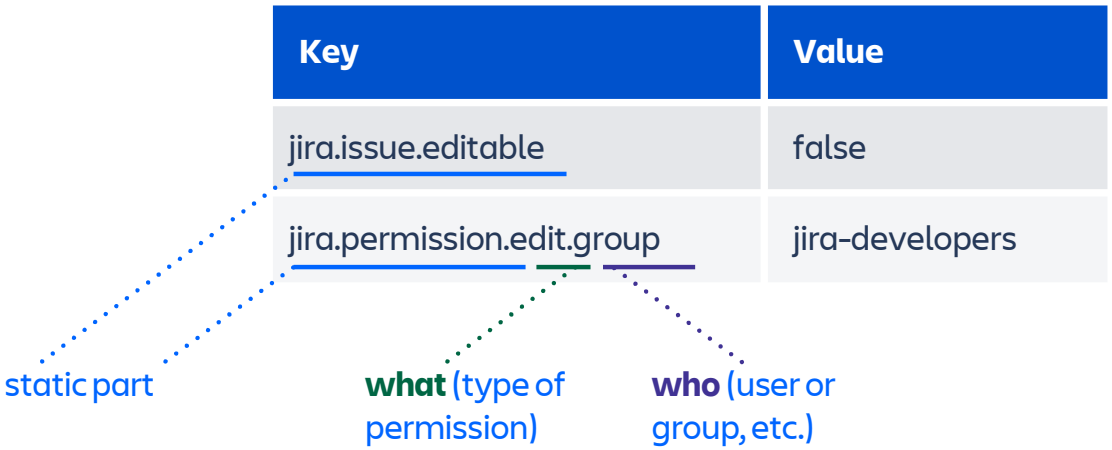
Resolutions in Jira are defined globally. Systems tend to grow over time, so do the number of possible resolutions. By default, if the resolution field is placed on a screen, all the resolutions will be visible. That might confuse users, as some resolutions might not be appropriate for a specific project. This property enables you to limit the resolutions on a workflow by workflow basis. In this example, there are nine resolutions defined in Jira administration. However, you only want five of these to be available to users when they resolve issues in your workflow.

Other examples are to show all available resolutions excepts Fixed. Or limit the available resolutions to Fixed and Won't Fix.

These properties apply once the resolution field is available on a transition.

How to set this property will be covered later in this module.

Property Key-Value Pairs



When creating a property, you enter the property key and the value, creating a key value pair.
Here you see two examples of property key-value pairs. The first one controls the editing of issues in a particular status. It's key is 'jira.issue.editable' and the value is 'false'. The second one controls what various users can do with an issue in a particular status. The is the most common use case for properties. This is a little more complicated. The static part is jira.permission. You specify what permission e.g. comment, edit, close etc. and who you're going to restrict it to i.e. group, user, assignee, reporter, lead, userCF custom field containing user), or projectrole. For the value you enter the actual group name, or username, etc.
You can create properties for either transitions or statuses.
See the next slide for details on setting properties and whether they apply to transitions or statuses.

Property Keys & Values

Purpose	Apply to	Key	Example Values	Input
Restrict permission	Status	jira.permission.<permission>.<type>	jira-administrators, agrant, 10002	Project role, group, user...
Make issue read-only	Status	jira.issue.editable	false	true or false
Define resolutions available	Transition	jira.field.resolution.include jira.field.resolution.exclude	1,2,3,4	Comma separated list of resolution IDs
Translate submit button	Transition	jira.i18n.submit	closeissue.close	i18n property key

Here you see the properties available to add to workflows.

- For the jira.permission key, you specify the permission e.g. comment, edit, close etc. and the type you're going to restrict it to i.e. group, user, assignee, reporter, lead, userCF custom field containing user), or projectrole. For example, to restrict edits to the Managers project role you'd use a key of jira.permission.edit.projectrole and a value of the project role id (not the name). For most others e.g. group, you use the name. To find out the project role id, mouse over the edit button for this project role on the Jira administration Project roles page.
- You set the 'editable' property to 'false' to prevent issues from being edited when they are in a particular workflow status.
- jira.field.resolution.include lets you specify which resolutions to include in a transition. Whereas jira.field.resolution.exclude lets you specify which resolutions to exclude in a transition. To set the value for these properties, you use the resolution IDs.
- Use i18n properties to translate the text of the buttons on screens for transitions. The value to be entered is the property that's being used in your language files. You need to have the relevant Jira language packs installed on your instance and you will need to make changes to the language jar files used by Jira server.

For more information, see https://confluence.atlassian.com/adminjiraserver/workflow-properties-938847526.html (Server/DC) or http://go.atlassian.com/cloud_wfprops (Cloud).

Common Mistakes

- Not checking if a transition is shared before adding a property
- Blank spaces in property keys
- Using too many properties

A common mistake is not checking if a transition is shared before adding a property. If transitions are being reused, they also share their properties. This is often the case with custom sequences using opsbar-sequence. If you intend to order transitions differently, you might need to create separate transitions rather than impacting other workflows.
You can only review properties by viewing the individual transition or status. If you're reusing transitions and statuses, setting properties on them may cause some unexpected behavior.
Blank spaces are no allowed in property keys.
Often too many properties are created. For example, adding multiple jira.edit.permission.user properties when you could use a single jira.edit.permission.group property. Think if a property is absolutely necessary before adding it. Use properties sparsely as they're hard to maintain and easy to confuse users or unexperienced admins. For example, the admin permission helpers don't apply when using permission properties even though permissions are being defined.

Best Practices

- Define project permissions first before using properties
- Instead of read-only status, limit to Jira admins so they can do bulk updates
- Test, test, test!

Define project permissions first before using properties. Properties refine project permissions but they don’t overwrite them! For example, you can’t grant a user the permission to edit an issue by defining a property. He has to have the Edit Issues project permission too.
Try not to make statuses "read-only”. Instead, make them only writable to Jira admins, that way they can still perform bulk updates if needed. If an issue can’t be edited, no bulk-updates will be possible.
Ensure you test your workflows thoroughly after adding properties. Mistakes aren't easily spotted. For example, when a group used in property has been deleted. Test properties in a development environment before going live so you can ensure that they work as expected.

See how it's done

- View a property in the Jira system workflow

View a property in the Jira system workflow

Are you getting it?

To ensure that closed issues can only be edited by project administrators, you would use a:

a. Post function
b. Property

To ensure that closed issues can only be edited by project administrators, you would use a:

a. Post function

b. Property

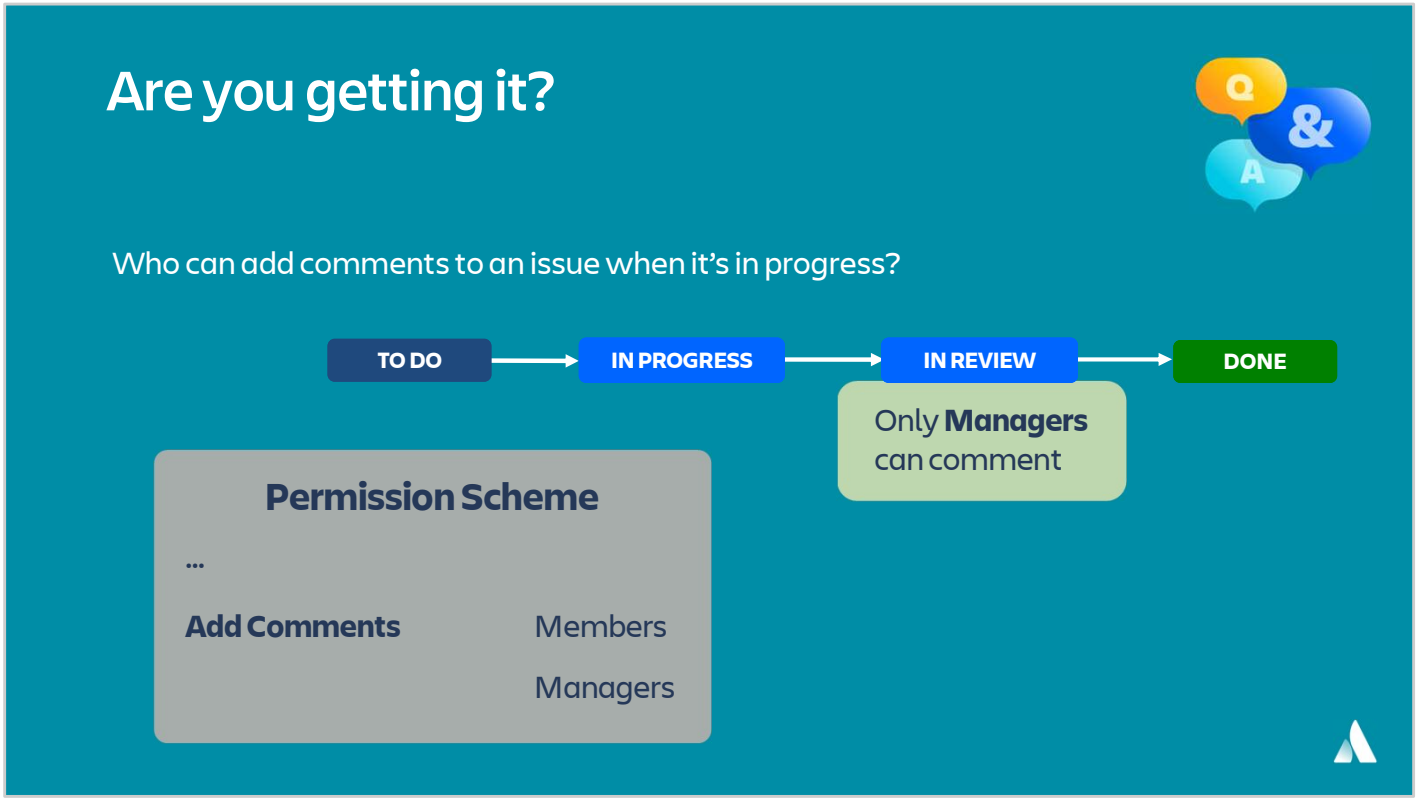
Did you get it?

To ensure that closed issues can only be edited by project administrators, you would use a:

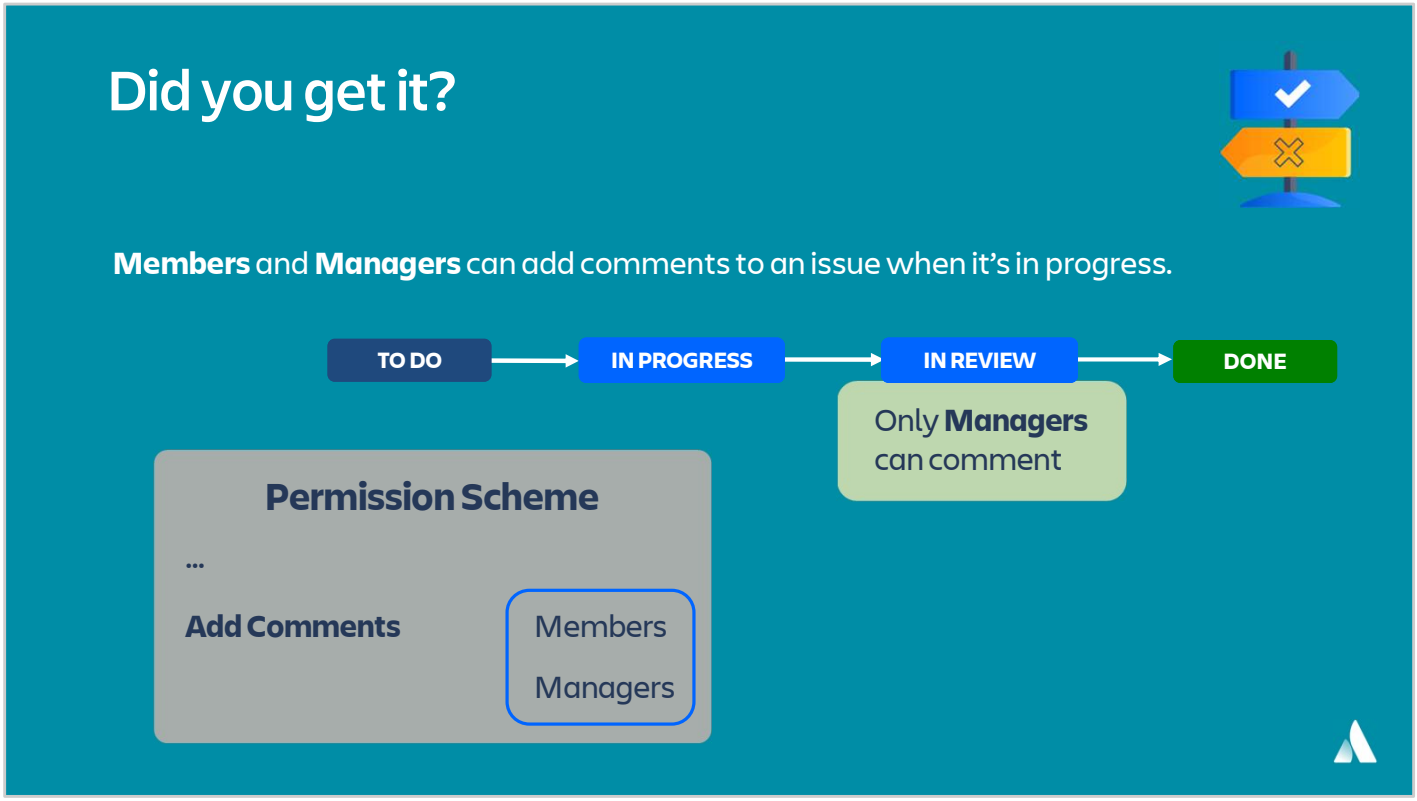
a. Post function

✓ b. Property

Answer: Editing an issue is controlled by permissions. Status based permissions can be controlled using properties.



Here you see the workflow used in the example. The permission scheme specifies that users who are in either the Members or Managers project role can add comments to issues. There is a property on the IN REVIEW status that specifies only users who are in the Managers project role can comment on issues.
The answer is on the next slide.



Answer: Users who are in the Members and Managers project roles can add comments to an issue when it's in the IN PROGRESS status. Only users in the Managers project role can add comments when it's in the IN REVIEW status.

Are you getting it?

If you wanted only Approvers to edit issues when they are in the Review status, which property would you use?

a. jira.issue.editable
b. jira.permission.edit.<...>

If you wanted only Approvers to edit issues when they are in the Review status, which property would you use?

a. jira.issue.editable

b. jira.permission.edit.<...>
(Approvers is a project role.)

Did you get it?

If you wanted only Approvers to edit issues when they are in the Review status, which property would you use?

a. jira.issue.editable

✓ b. jira.permission.edit.<...>

Answer: b. To allow only members of the Approvers project role to edit issues when they're in the Review status, you'd use the jira.permission property. This allows you to specify the permission, in this case edit, as well as who it's restricted to, in this case Approvers. The jira.issue.editable property is used to make issues read-only for everyone.

Are you getting it?

Which of these are best practices?

a. Define project permissions first before using properties
b. Always make closed issues read-only
c. Use project roles rather than individual users in permission properties
d. Check whether a transition is shared before adding a property

Which of these are best practices?

a. Define project permissions first before using properties

b. Always make closed issues read-only

c. Use project roles rather than individual users in permission properties

d. Check whether a transition is shared before adding a property

Did you get it?

Which of these are best practices?

- ✓ a. Define project permissions first before using properties
- b. Always make closed issues read-only
- ✓ c. Use project roles rather than individual users in permission properties
- ✓ d. Check whether a transition is shared before adding a property

Answer: a, c, and d. b is not a best practice because when you make an issue read-only no-one can edit that issue. Even with closed issues the Jira admin may want to perform some edits, often with bulk edit. So it's better to restrict the edit permission to just Jira admins rather than make it read-only.

Takeaways

- Check whether transitions are shared before adding properties
- Use properties sparsely as they’re hard to maintain
- Test, test, test!

Check whether transitions are shared before adding properties to avoid unintended consequences.
Use properties sparsely as they’re hard to maintain and easily confuse inexperienced admins.
Always test your properties on a development environment to ensure they behave as expected.

Lab 6 – Extending Workflows with Properties

- Exercise 1
  - Restrict who can delete and edit closed issues
- Exercise 2
  - View properties in Jira workflow

7 Taking It to the Next Level

Course Overview

Covering the Basics

Creating Conditions & Validators

Automating Your Workflows with Post Functions

Triggering Transitions

Extending Workflows with Properties

Taking It to the Next Level

What will you learn?

- Translate requirements into workflows
- Edit workflows using advanced workflow techniques
- Deal with challenges when customizing workflows
- Share workflows

In this module we'll go beyond the basics of editing workflows. We'll take a step back and look at the bigger picture of creating and managing workflows. Here you'll learn how to translate requirements into workflows, edit workflows using more advanced workflow techniques, deal with challenges when customizing workflows, share workflows, and more.

Goals of Business Analysis

Build it right the first time

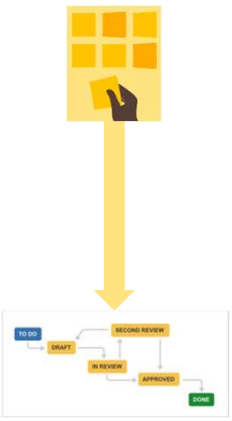
Reduce future overhead

Design it well so people want to use it

Help people get their work done quickly and efficiently

These are the goals of business analysis.
If you build it right the first time it will reduce future overhead. Plan in advance (measure twice, cut once). Understand requirements before you start building so you don't have to backtrack or undo changes. For example, If you add in the wrong statuses and have to migrate data later to new statuses, it is a nuisance and wastes time.
Design workflows well so people want to use them and they help people get their work done more quickly and efficiently. Build a solution that doesn't become work itself, but instead gets out of the way and let's people focus on their actual job tasks.

Approach



1. Gather the business requirements
2. Identify key requirements
3. Translate requirements & map to Jira configurations
4. Implement in stage and review
5. Implement in production

To create a successful Jira workflow, it needs to relate to the business where it is being used. It could be the case that you need a completely new workflow or you're updating a basic workflow as you need something more complex. Actual implementation in Jira can be fast. 90% of the effort is discussing and mapping out the business processes, etc.
This approach involves a number of steps:

1. Gathering the business requirements from stakeholders
2. Breaking down the business requirements to identify the key requirements
3. Translating those requirements and finding the best implementation option in Jira. Here we map the requirements to Jira configurations
4. Implement the workflow in your staging instance then go through a design review
5. Finally implement the workflow in your production instance

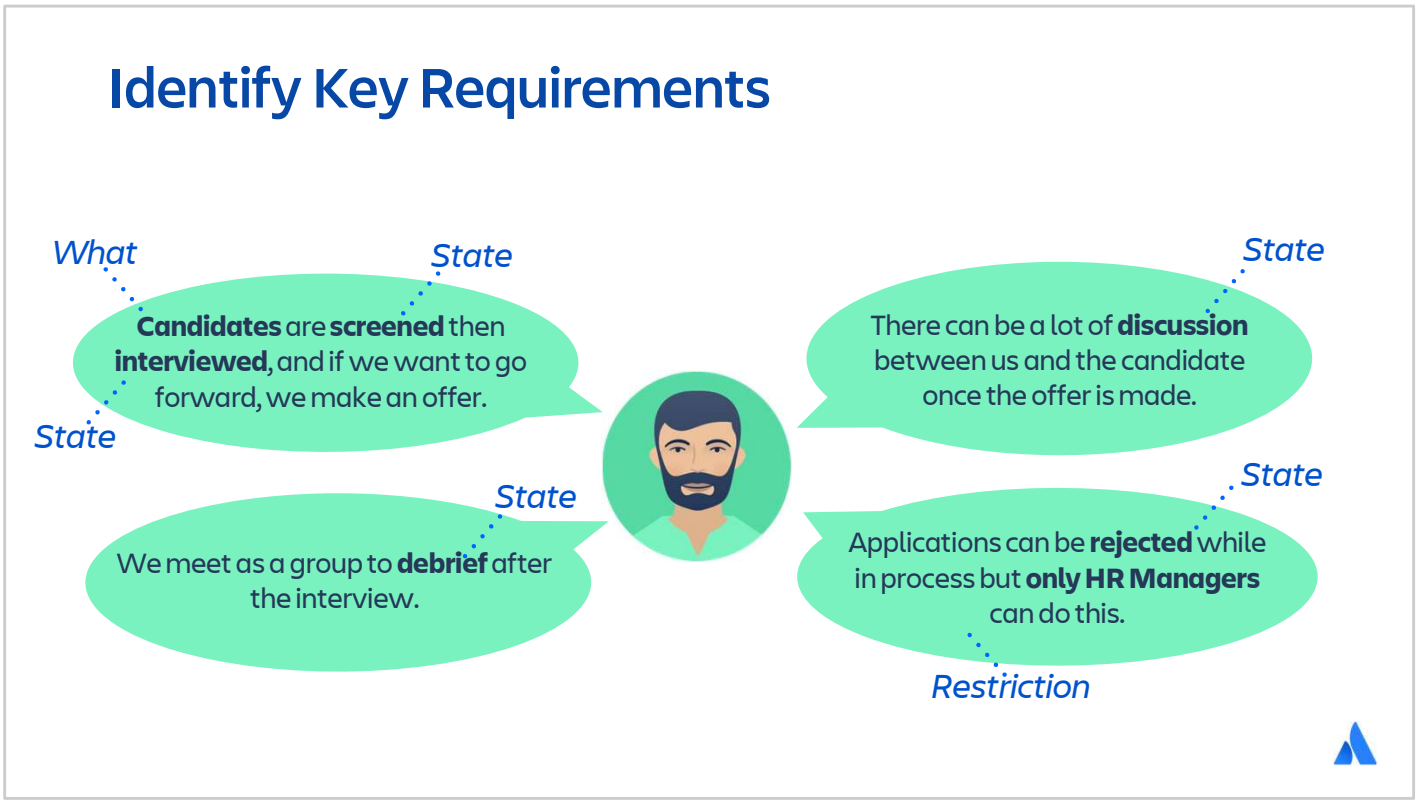
Gathering Requirements

Get together team stakeholders and ask:

- Who creates/submits? And why?
- Where does it go next?
- Who performs the work?
- Who approves?
- Why do you do it this way?
- What are the dependencies?
- etc.

Before creating or customizing a workflow, admins should get together team stakeholders and gather their requirements. This is as simple as leading a meeting and having the team write out their work processes on a whiteboard. It's critical to understand your team's needs before translating them into a workflow.
Talk through their process. Ask them "Why do you do it this way?" You'll often find ways to change.
Don't necessarily set out to 'make a workflow,' rather document their actual process. Mentally or physically, be a work item and document each step. Who creates/submits, why, where does it go next, who approves, who performs the work, what are the dependencies, etc.
During meetings in addition to white boarding and Post It notes, you can use flow charts, drawings/diagrams of fields and screens, etc.

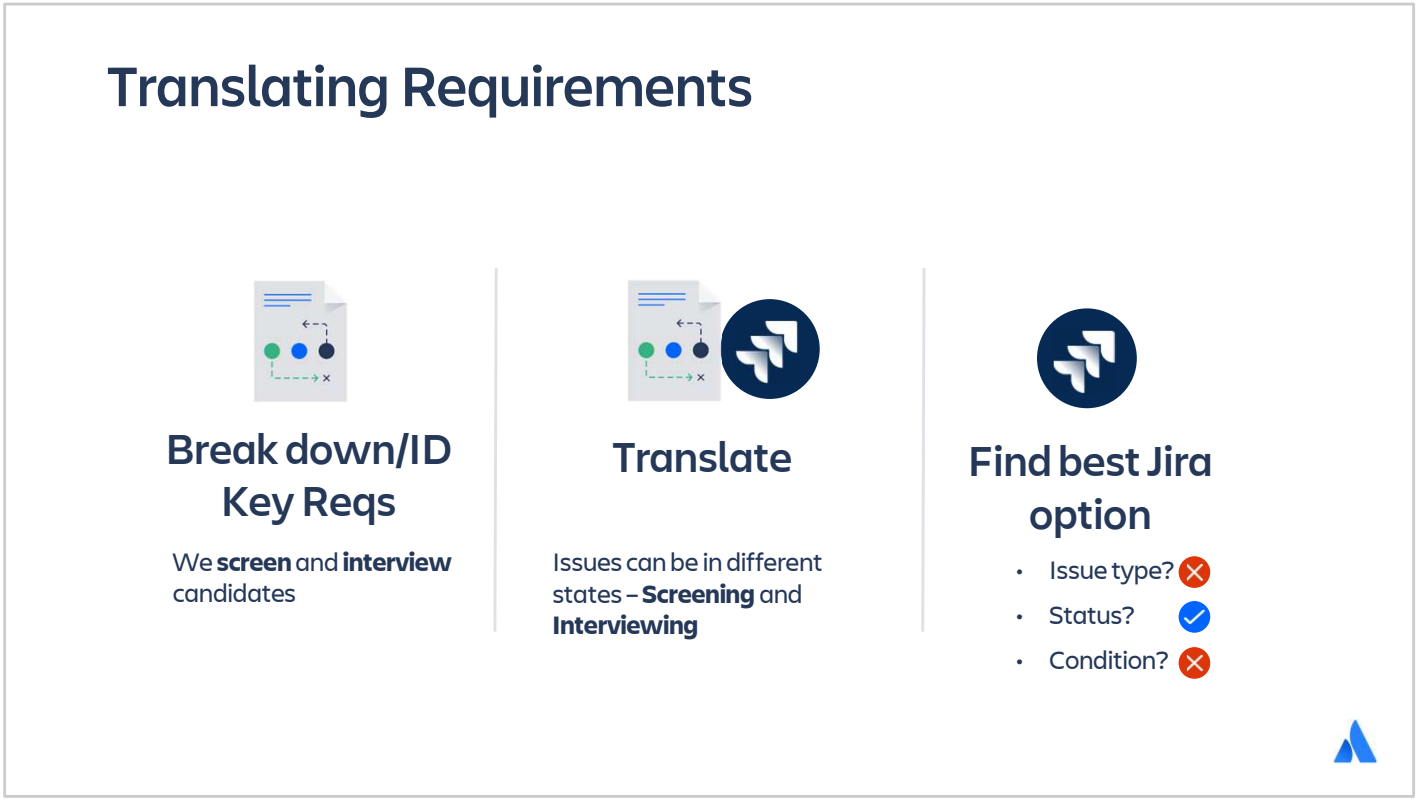
This is an example of some requirements from HR for a recruitment workflow. HR wants a workflow to track job candidates from application to offer (or rejection). These are the requirements that came out of the meeting with the stakeholders.



Requirements are typically in business language as your typical Jira user doesn't know how to implement workflows in Jira. After you've gathered the requirements, you need to focus on the key parts of them. This helps you to focus in on what would need to be implemented in Jira.

Focus on key parts of each requirement. In this example, we can divide them into different parts:

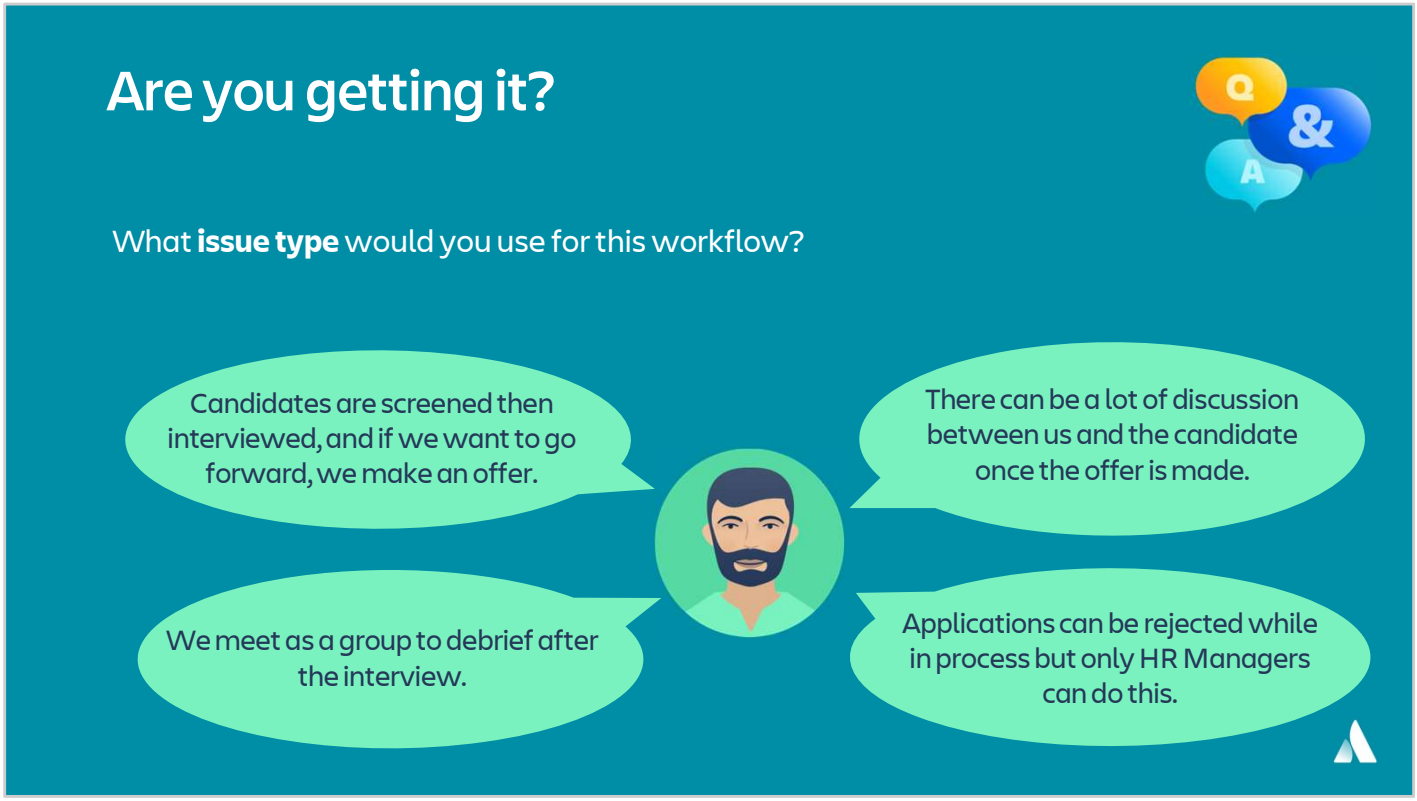
1. Identify what the item is that will be moving through the workflow
2. Identify the states that this item can be in
3. Identify any restrictions on who can perform a certain action



This process can be re-used every time there's new requirements. After you've identified the key requirements, follow this process for each requirement separately:

1. Split them up, break them down into pieces that can be addressed in Jira.
2. Think about how those "mini requirements" relate to your Jira instance. Here we see an example of one of the mini requirements from our HR Recruitment example. We need to have issues go through different states in the workflow – Screening and Interviewing.
3. How do I implement different states in a workflow? By creating a new status for each of these states (or re-using an existing one). Plan out what changes you will make in Jira before you actually implement them.

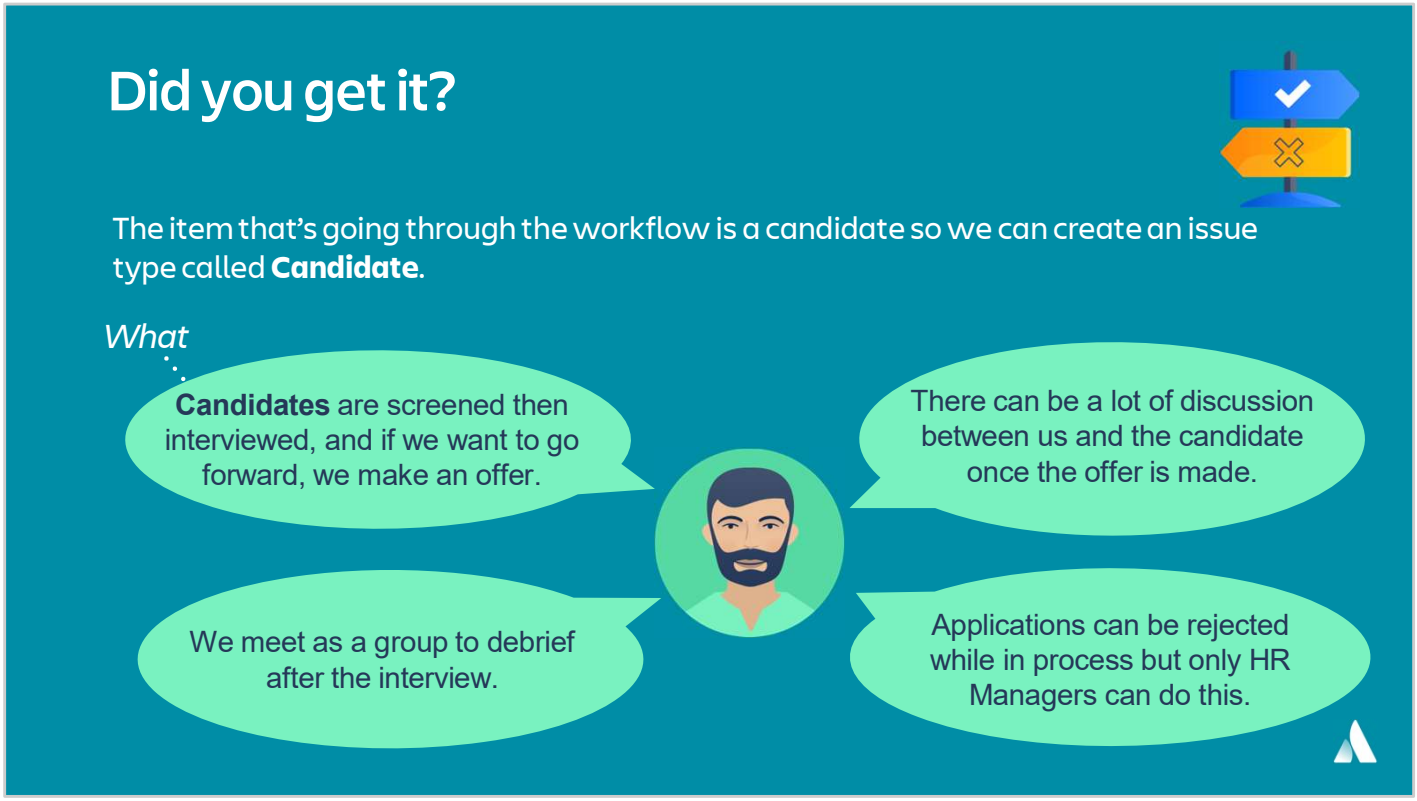
For example, from your stakeholder meetings you may now have notes, post its and diagrams that lay out the business processes that teams use to take their issues from creation through to completion. Now you formalize all this information, for example, into a UML or process diagram. This gives you a clear picture of what you need to implement in Jira and includes all the workflow statuses, transitions, conditions, validators, post functions, etc. for each workflow. You also know at this point which workflow needs to apply to which development issue types – one workflow for bugs, stories and feature request, and another workflow for all the other issue types. When you take the time to map out your process, it helps you anticipate
potential roadblocks and lets you build safeguards into your timeline.
Define a workflow that follows best practices and facilitates your process. Use states (statuses) and paths (transitions) that make sense and minimize confusion.



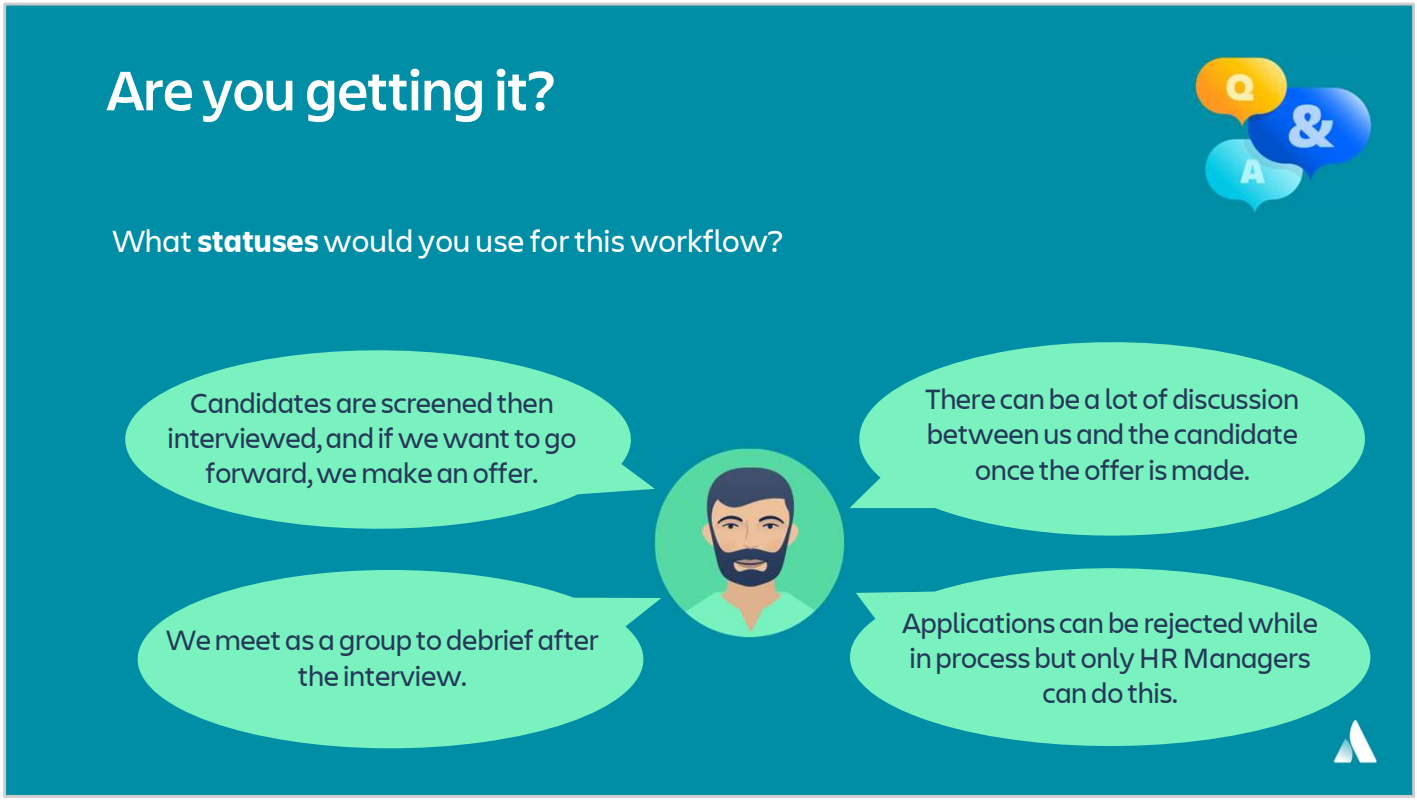
Identify some more key requirements and translate them into Jira configurations.

1. Candidates are screened then interviewed, and if we want to go forward, we make an offer.
2. We meet as a group to debrief after the interview.
3. There can be a lot of discussion between us and the candidate once the offer is made.
4. Applications can be rejected while in process but only HR Managers can do this.

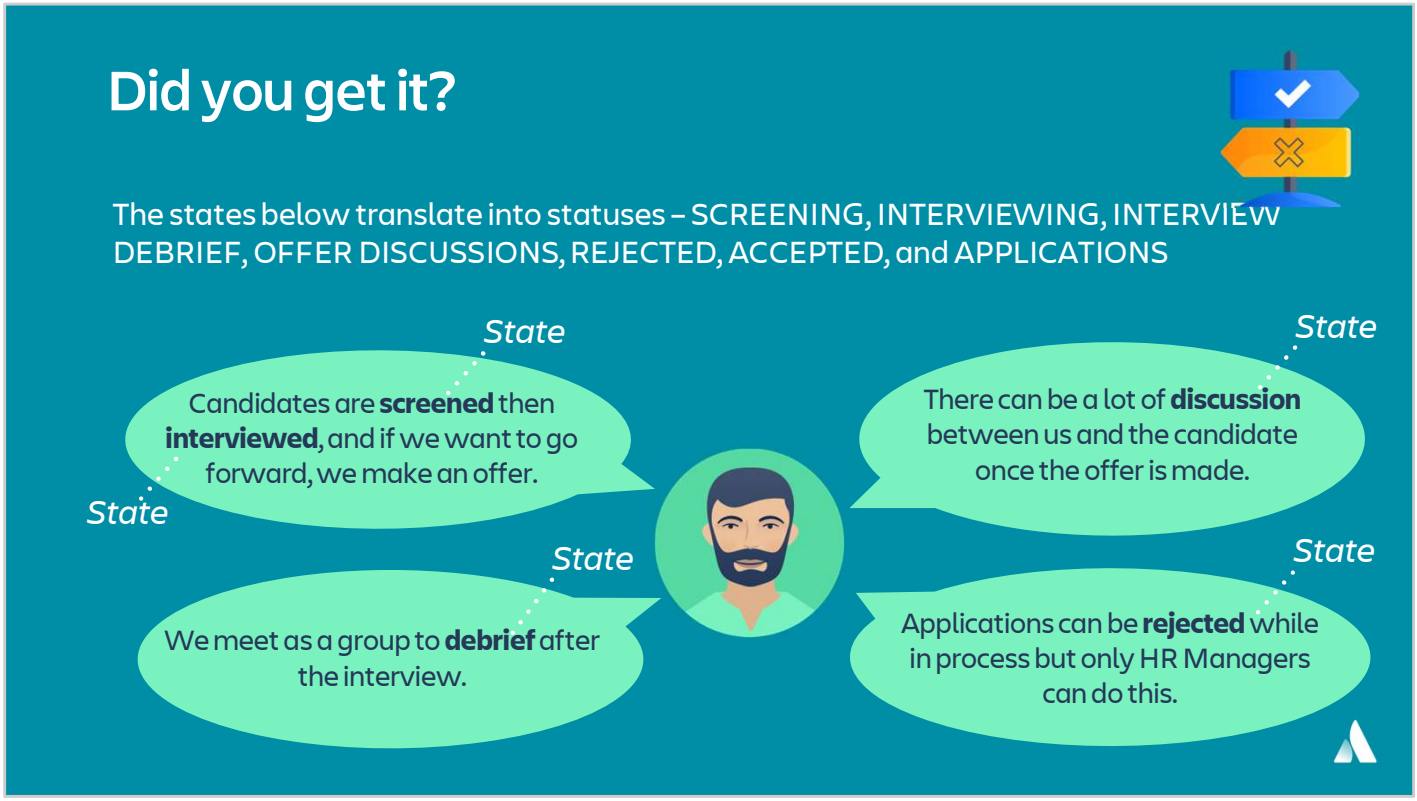
What **issue type** would you use for this workflow?



Answer: We need to use an issue type for the item that's going through the workflow. In this case it's a candidate, so we can create an issue type called Candidate to use (if one doesn't already exist). We could also look at the other issue types that exist, perhaps there's one that could be reused, for example, Person.



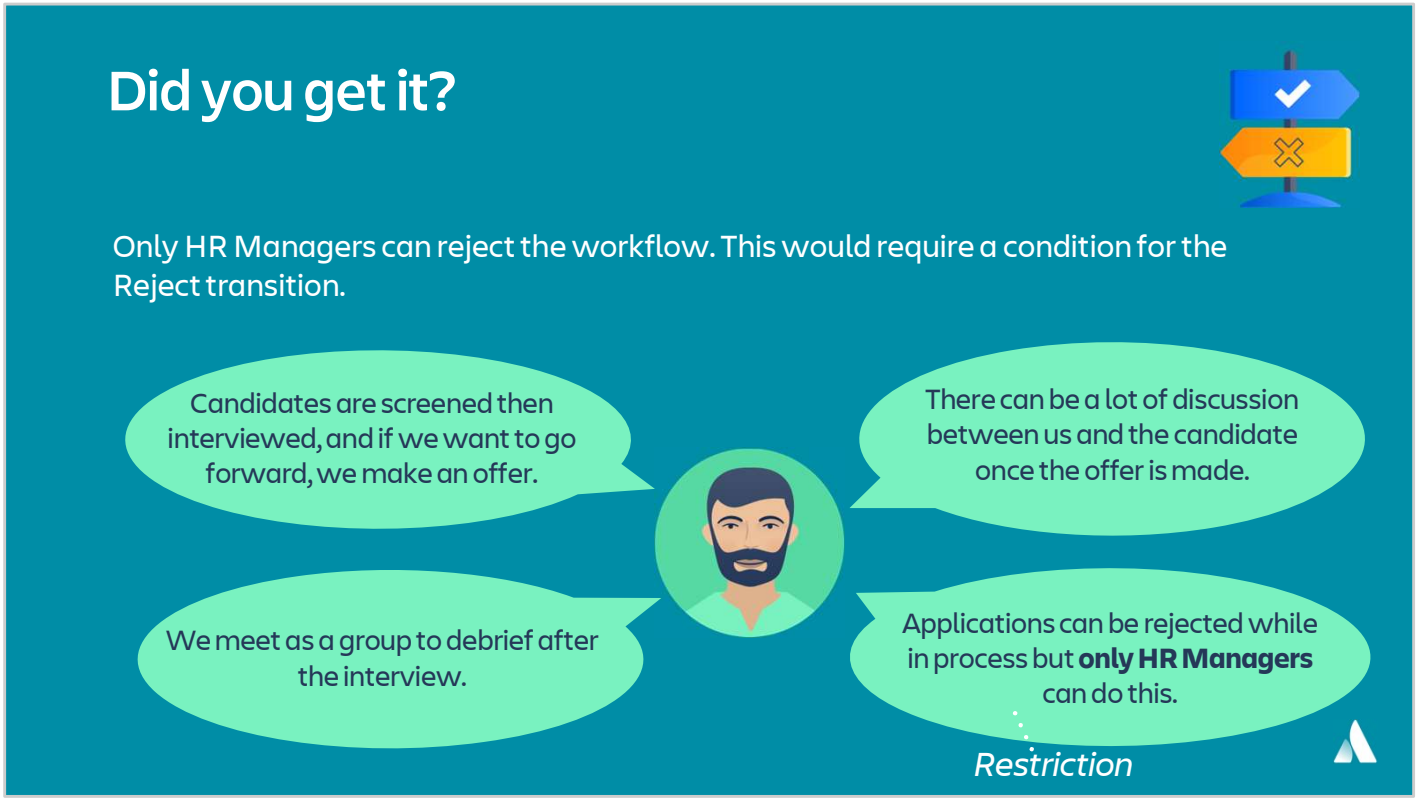
What **statuses** would you use for this workflow?



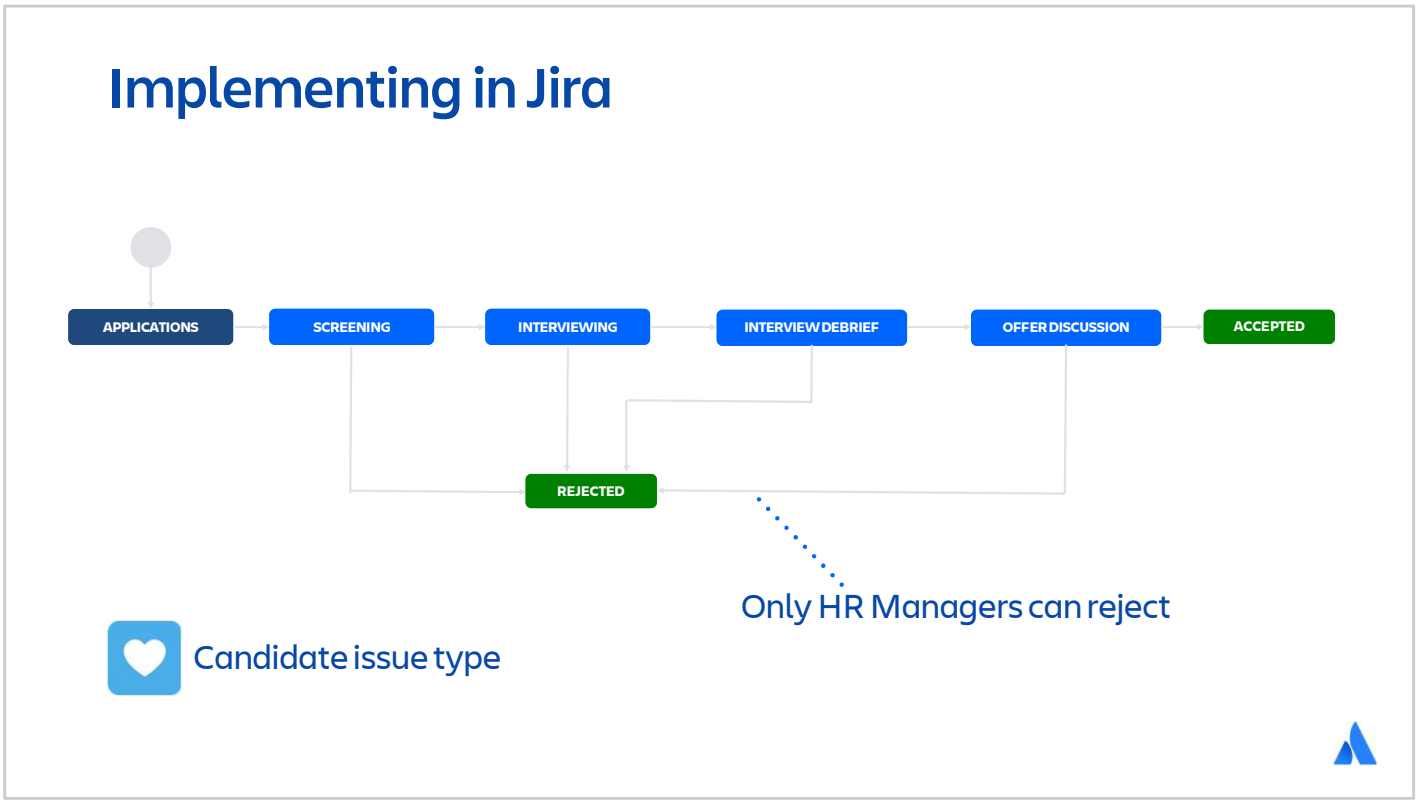
Answer: First we'll look at what states the issues can be in. We can break down the requirements and identify the SCREENING, INTERVIEWING, INTERVIEW DEBRIEF, OFFER DISCUSSIONS, and REJECTED states which can be used as workflow statuses. You could come up with different status names but these are close to what's needed. These are all In Progress statuses.

Also we need a done (final) state other than REJECTED which would be used if the offer was accepted i.e. ACCEPTED.
Also we need a To Do (initial) state that applications are in before they are being screened (in progress). This could be TO DO, but we'll simply call it APPLICATIONS.

Is there anything else required for this workflow?



Answer: As only HR Managers can reject a candidate, you'd need to add a condition to the Reject transition that only users in the HR Managers role (or group if you'd set them up that way) can execute this transition.



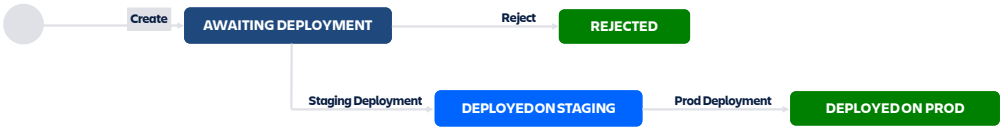
After gathering requirements, identifying the key requirements and translating them, implement them in Jira. You create your workflows in Jira creating all the workflow statuses, transitions, conditions, validators, post functions, etc. for each workflow. And associating workflows with an issue type or types by creating workflow schemes.
Here's what the workflow could look like for the HR Recruitment example. It has seven statuses which is about the limit you'd want for most workflows. Here are the statuses broken down by category:

- To Do - APPLICATIONS
- In Progress – SCREENING, INTERVIEWING, INTERVIEW DEBRIEF, OFFER DISCUSSINS
- Done – ACCEPTED, REJECTED

The issue type used is Candidate.
There's a condition on the Reject transition (which is used four times), that only users in the HR Managers role can execute this transition.
The workflow you see here is the workflow that's provided with the Recruitment project template on Jira Core for the Cloud. To find out more about Jira Core workflows, see https://www.atlassian.com/blog/jira-core/how-to-set-up-business-workflows-in-jira-core (Server/DC) or https://support.atlassian.com/jira-work-management/docs/how-to-create-workflows/ (Cloud).

Approval Requirement Example

Requirement Deployment for both Staging and Production now needs to be approved by a Release Manager



How would you break down & translate this requirement?

Let's look at another example of a requirement. In this case we have an existing Deployment Tracker workflow. There's a new requirement that deployment has to be approved for both Staging and Production. And further that approval can only be done by Release Managers.
How would you break down this requirement? See the next slide.

Translating Requirements

Break down/ID Key Reqs

1. Staging & Production need to be approved
2. Approval can only be done by a Release Manager

Translate

1. Add approval to both Staging & Production
2. Only members of Release Managers role can approve

Find best Jira option

1. New approval step or transition?
2. Condition

A. Break down/ID Key Requirements

1. Staging & Production need to be approved
2. Approval can only be done by a Release Manager.

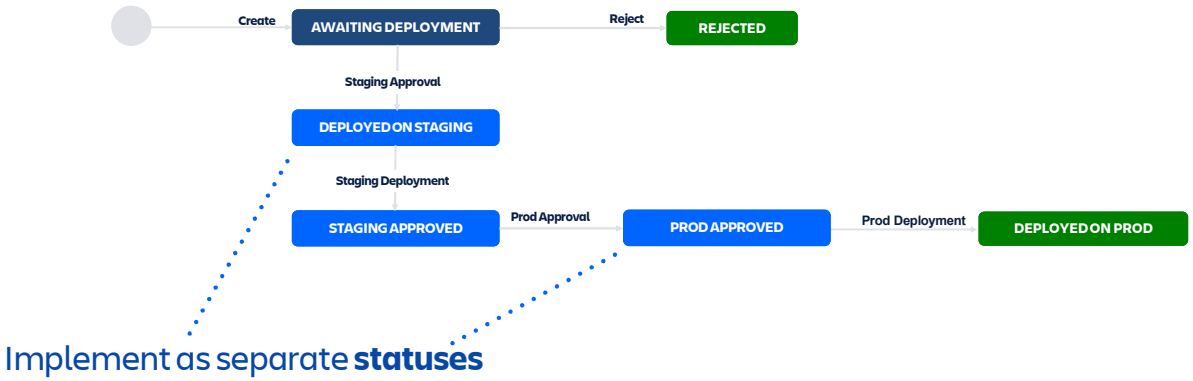
B. **Translate**

1. Add approval to both Staging & Production
2. Only members of Release Managers role can approve

C. **Find best Jira option**

1. New approval step or transition?
2. Condition

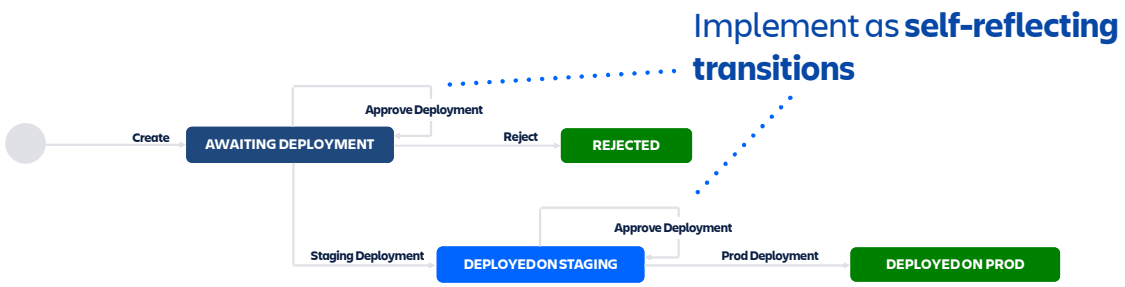
Implementing Approvals Option 1



Implement as separate **statuses**

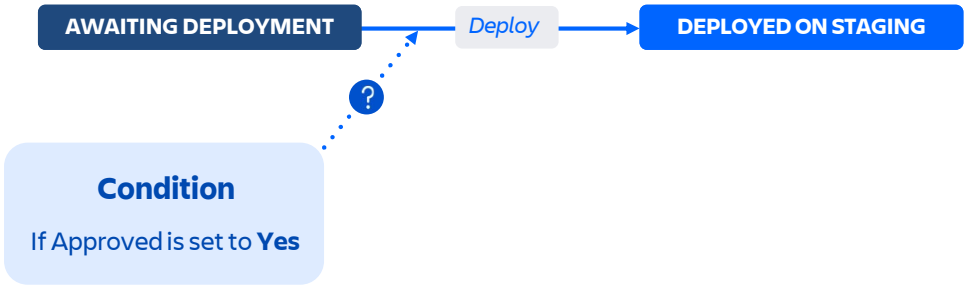
There are a few ways you can add approvals to the Deployment workflow. One option is to implement the approvals as separate statuses, for example, STAGING APPROVED and PROD APPROVED. You'd also need to add the transitions going to these statuses, Staging Approval and Prod Approval. This works well for many workflows especially if they're fairly simple to start with. However, with more complex workflows this adds more complexity and makes them more difficult to maintain.

Implementing Approvals Option 2



Transitions don‘t have to change the status of an issue. They can point to the current status an issue is in.
When designing a workflow, it's a best practice to keep the workflow as simple as possible, so consider if a transition or status is actually needed. One way to achieve reuse is to use self-reflecting transitions (instead of having multiple statuses within your workflow).
In this example we implement the approvals requirement by using self-reflecting transitions. We have a set of self-reflecting approval transitions that go from a deployment status back to the same status, rather than having a transition from each deployment status to an approval status. The reason that a self-reflecting transition is better practice is that it cuts down the number of statuses with ingoing and outgoing transitions, making workflows less complex and easier to manage. These work well in combination with flags. We’ll discuss those in the next slide.

Using Flags Instead of Statuses



Sometimes you may want to use a field instead of a status to indicate a state the issue is in. In the approval example, we want to keep the workflow simple so instead of adding approval statuses, we use self-reflecting transitions and add an Approved field to the issue. This can be unset (not approved) or set to Yes (approved). This flag can then be used in workflows conditions, post functions, etc.

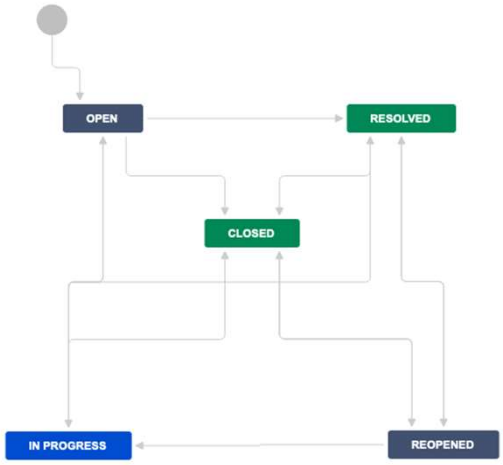
A flag is a field that gets set with a value. You do this to mark an issue for easy filtering on certain criteria or to restrict issues of a certain criteria from being transitioned. Usually a flag is a field, on an issue, of select list type. A user selects a value such as yes from a drop-down. However, Jira does not provide the ability to allow only certain users to update a field (field level security). So, in order to control how a field gets updated, you should look to use post functions.

Typically within Jira, flags are custom fields created by users. Thus, for Jira Server, you need to use the Update Issue Custom Field post function (provided by Suite Utilities for Jira) to update these. Out of the box, Jira Server provides the Update Issue Field post function that only updates system fields and not custom fields.

Building New Workflows

1. Copy and customize one of Jira's non-simplified default workflows
2. Copy and customize an existing custom workflow
3. Create a new workflow from scratch

Jira System Workflow



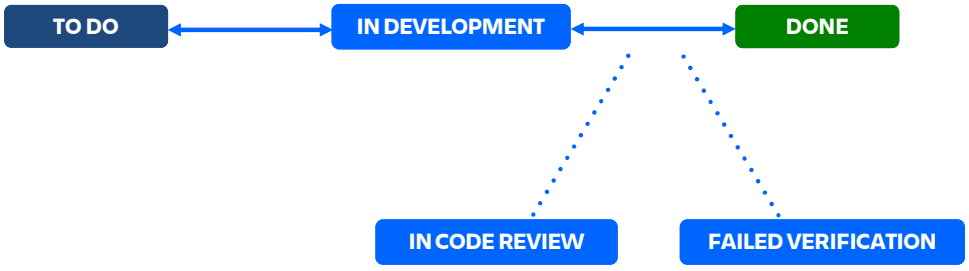
If you need to go beyond the capabilities of the default workflow you got with a project, you can build your own workflow in three different ways:

- Copy and customize one of the workflows that come with Jira. For example, you could customize the jira (Read-only System Workflow) that's pictured here. Or you could customize the workflow that came with a project template, for example Process Management (Jira Core). But don't start with any of the default workflows that are Simplified Workflows. These shouldn't be modified.
- Copy and customize an already existing workflow that you or someone else has already customized. This can include importing a workflow from the Atlassian Marketplace or another instance. We'll cover importing and exporting workflows later in this module.
- Create a new workflow from scratch.

Copying and customizing an existing workflow (either one of the defaults or any already existing customized workflow) is usually the easiest way to create a new workflow. Look for an existing workflow that's close to the workflow you want to create.

Creating a workflow from scratch could be a lot of work. It's easy to forget things when you're creating everything from scratch. For example, setting the resolution, updating the event to a close event on final transition, etc.

Adding New Statuses



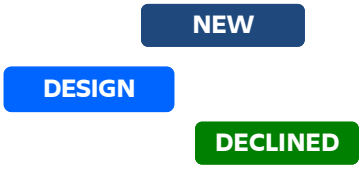
Would it make sense to add these two new statuses?

Stakeholders often want to have statuses for each part of the workflow. That's generally a good thing, but remember: each status adds more transitions and complexity. Aim for simple and scalable instead. Whenever adding a new status to a workflow, make sure you have no other option. Let's look at two examples.

- Code review is an important part of the software development process. Jane, the development manager, wants to add a specific status called Code Review so it's clear to her team which issues are under active development, and which issues are awaiting review. Reviewing code is distinctly different than writing code, so it makes sense to add a new state.
- Bill, the QA manager, wants to add new status called Failed Verification for all issues that don't pass review by his team. I'd advise against doing this, as the test engineers can simply send any issue that fails review back to a previous state, such as In Development.

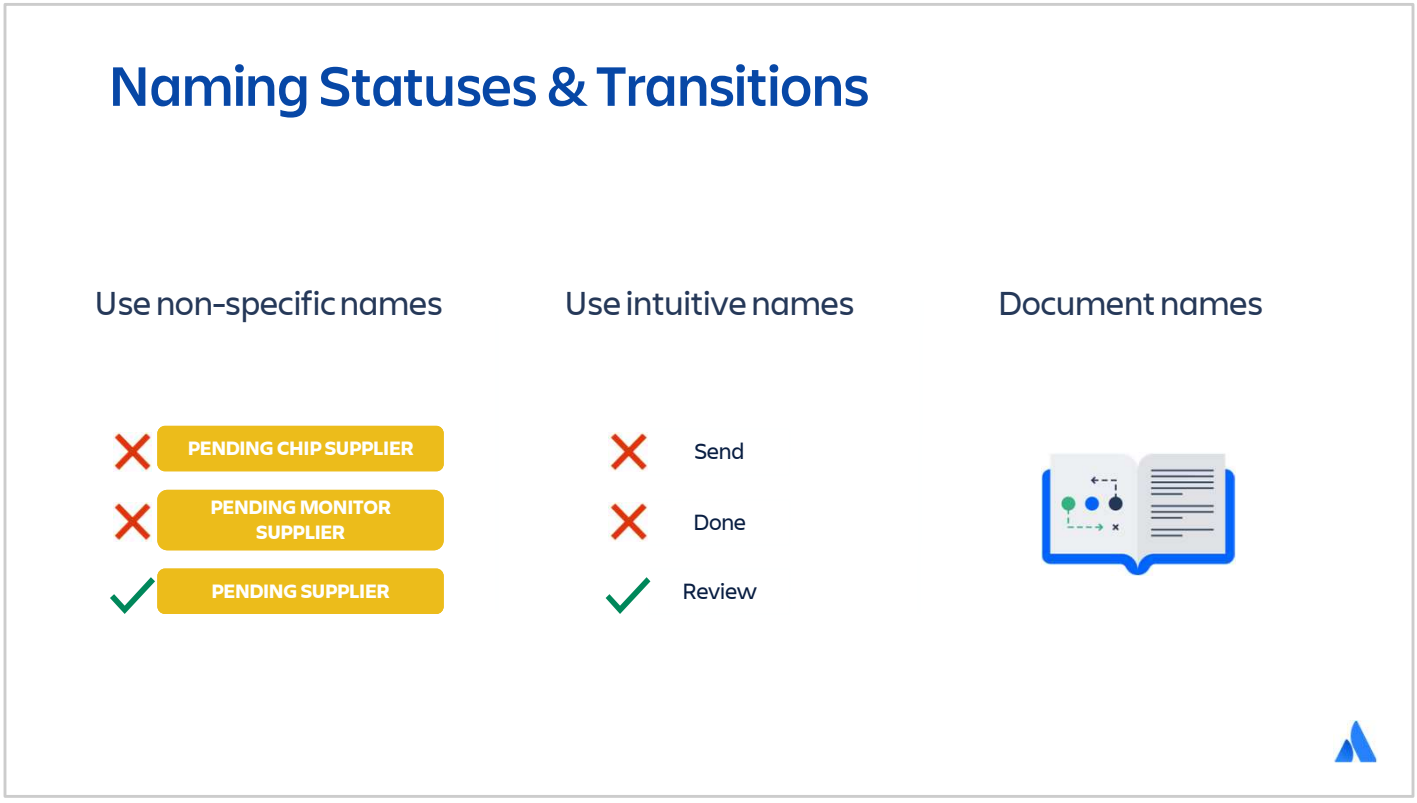
Don't exceed around 7 statuses unless it's complex change management, or a shared workflow where different phases will be used by different teams.

Creating New Statuses



- Start simple and add statuses only when you need them
- Create a status whenever:
  - There's a handoff from one person to another
  - A piece of work is going to be in that same status for a long time
- Question requests for new statuses
- Don’t create a lot of statuses

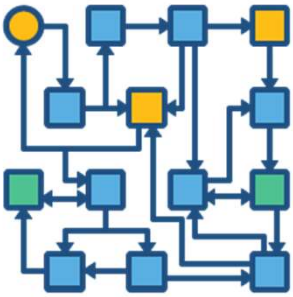
When editing workflows, you can use the default statuses that Jira ships with or create your own. Don’t over-complicate your workflow. You want to get fine-grained visibility into the status of work but building a workflow with 20 or 30 statuses results in a workflow nobody wants to use. It’s better to start with a simple workflow and add statuses when you need them. Make workflows that people like to use. When people enjoy using them, they’re more likely to keep the issue updated, which means your data is more accurate and actually useful.
Create statuses that reflect how people actually do their work. Create a new status is when work needs to be re-assigned to another person. Or when a piece of work is going to be in the same status for a significant period of time. This period of time will differ depending on the workplace and the overall period needed for the workflow to be completed.
Often management will want to track more data for reporting. It’s important to think through a request for more data. Will that data help us make better decisions? Enable us to help our customers better? Will the proposed change be the best way to get that data, or is there a better way we should think about? Don’t gather data for the sake of data. That will slow down work and negatively impact your team outcomes.
Don’t exceed around 7 statuses unless it’s complex change management, or a shared workflow where different phases will be used by different teams.
Only Jira administrators can create new statuses on the Statuses Jira administration page.



It's important to give statuses and transitions names that are clearly understood by all. Give statuses generic (non-specific) names so they can be shared. For example, rather than 'Pending Chip Supplier' and 'Pending Monitor Supplier', use 'Pending Supplier'.
Transition and status names should be intuitive. Users should never have to wonder which workflow transition to use next. For example, in a content management workflow, if the writer has finished their draft and needs to have it reviewed, simply use 'Review' rather than 'Send' or 'Done' which could be confusing. 'Send' could be interpreted as 'Send for Review' or 'Send for Publication' or some other send. 'Done' is confusing because this implies everything is done with the issue, which it is not. If someone looks at a status and is confused about what work is being performed in real life at that time, that indicates that better naming or more statuses is needed.
It's also good to document transition and status names, (use Confluence), to define the statuses and transitions. Include documentation of transition behaviours, transition properties, and status properties. Documentation facilitates both training and collaboration without having to provide direct access to Jira workflow.

Challenges When Customizing Workflows

- Overly complex workflows that nobody wants to use
- Users stuck on statuses that don't transition anywhere
- Too many statuses

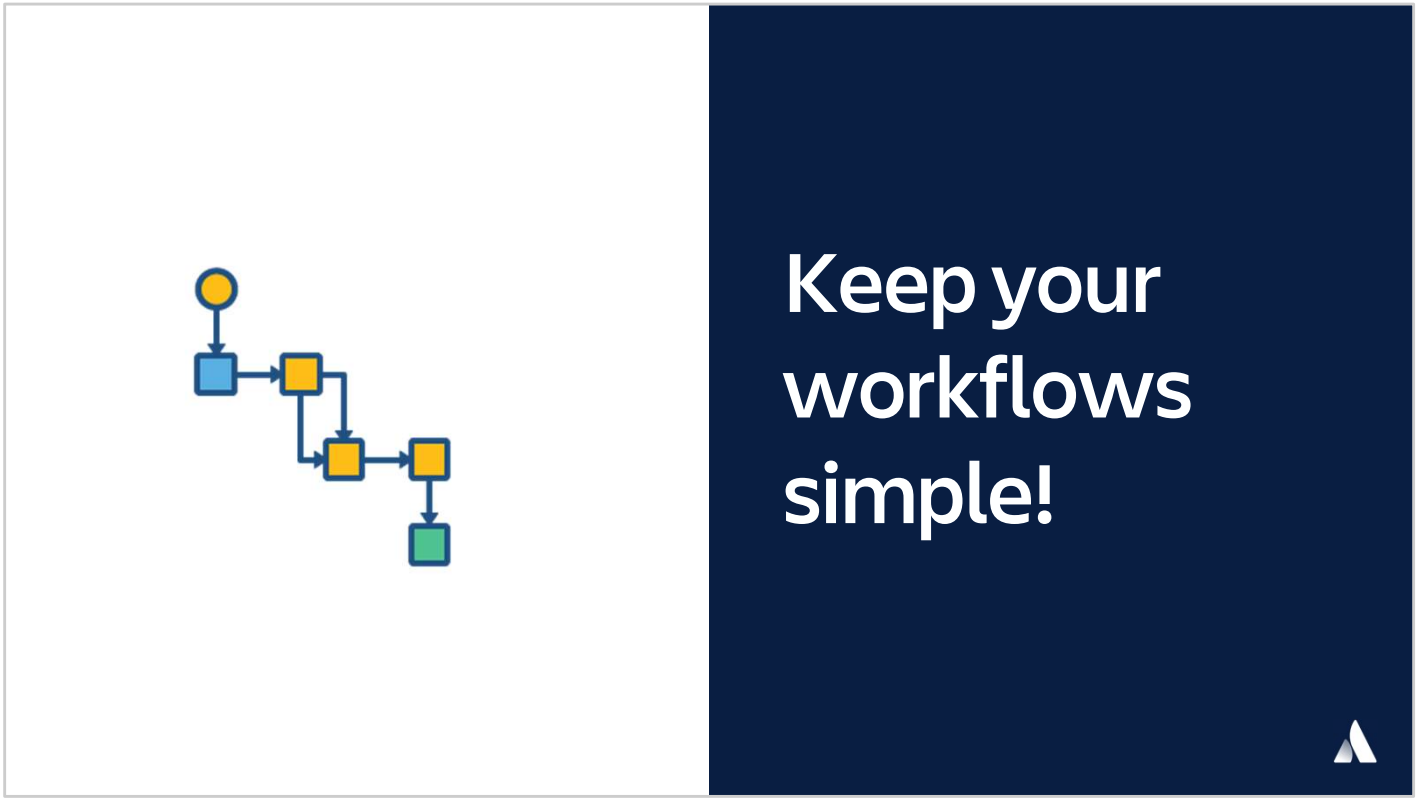


Do you have workflows that are so complex nobody wants to use them?
Do you have people stuck on statuses that don't transition anywhere?
Do you have far too many statuses in your workflow?
In Jira there is some workflow validation that the system does to identify statuses that are missing transitions, and also validate transition permission conditions when importing workflows.

Creating Workflow Best Practices

- Identify your key requirements and translate into Jira, before implementing
- Copy and customize an existing workflow rather than creating from scratch
- Keep the number of statuses and transitions to a minimum
- Ensure users don't get stuck on a status
- Use intuitive transition and status names
- Use screens sparingly

Keep your workflow simple! Too many statuses and thus too many options of available workflow transitions will confuse the user.
Stakeholders often want to have statuses for each part of the workflow. That's generally a good thing but remember: each status adds more transitions and complexity. Aim for simple and scalable instead. Whenever adding a new status to a workflow, make sure you have no other option.
For example, Bill, the QA manager, wants to add new status called 'Failed Verification' for all issues that don't pass review by his team. I'd advise against doing this, as the test engineers can simply send any issue that fails review back to a previous state, such as 'In Progress' or 'In Development'.
See http://blogs.atlassian.com/2013/10/building-workflow-awesome/.
Use transition screens sparingly, only when you need to get required information from the user, for example, a resolution.



Keep your workflows simple!

Avoid workflow sprawl

Teams in Space dev workflow
Mobile app workflow
SW workflow
Dev workflow
iOS workflow
Android app workflow
Project X workflow
SW development workflow
...

Use a small set of standard workflows for each function

SW Dev workflow
SW simplified workflow
Kanban workflow

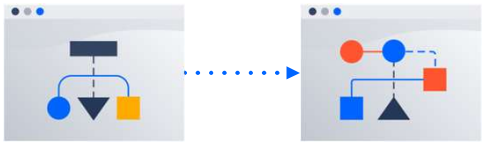
Aim to use a small set of standard workflows for each function. For example, there's no need to have twelve software development workflows when most of your teams follow the same process. In this example we see that all the workflows on the left could be replaced by three workflows – a standard SW Dev workflow that nearly all teams follow, a SW simplified workflow for very simple dev projects, and a Kanban workflow for Kanban projects.

You want to avoid workflow sprawl. It's hard to manage and update that many workflows and it's challenging at best to roll up stats when each team is doing something different.

Note however that if you share workflows, project admins won't be able to edit these workflows. They can only edit workflows that are not shared with any other projects. If you've set up a standard workflow, you wouldn't want project admins editing it anyway. They can still edit the ones on those projects that follow non-standard workflows.

Changing Workflows in Production

- ✓ Copy the workflow
- ✓ Version it and check name
- ✓ Document what changed
- ✓ Make your changes
- ✓ Test
- ✓ Replace the production version



It’s risky to edit an active workflow that’s in production. Once you publish it, the changes are live and if you made a mistake the consequences could be bad. Instead, copy the workflow, version it, ensure that it's name matches your artifact naming convention, make a note of what changed, then make your changes and save. Replace the current version with the new one after testing.

Testing Your Workflows

- ✓ Go through each status & transition
- ✓ Check screens
- ✓ Check conditions
- ✓ Confirm there are no validation alerts
- ✓ Build and test on a test instance
- ✓ Use a test project on a production instance

Once you've published your new workflow, test out an issue, make sure that it works as expected. Check status, transitions, screen, conditions, etc. You may need to do that with virtual users that have different permissions, roles, and groups.
An alternative to working live, and preferred when working in a staging/production environment, is to build the workflow on your test instance and test it there too. Then you can export the workflow, import the workflow to production, and test again. Export and importing workflows is covered on the next slide.
If you need to create and test your workflow on your production environment then create a test project and test there.

Are you getting it?

Which of these are recommended best practices:

a. Keep your workflows simple
b. Create new workflows from scratch
c. Use a screen for each transition
d. Edit copies of production workflows rather than editing active workflows

Which of these are recommended best practices?

a. Keep your workflows simple

b. Create new workflows from scratch

c. Use a screen for each transition

d. Edit copies of production workflows rather than editing active workflows

Did you get it?

Which of these are recommended best practices:

- ✓ a. Keep your workflows simple
- b. Create new workflows from scratch
- c. Use a screen for each transition
- ✓ d. Edit copies of production workflows rather than editing active workflows

Answer: a and d

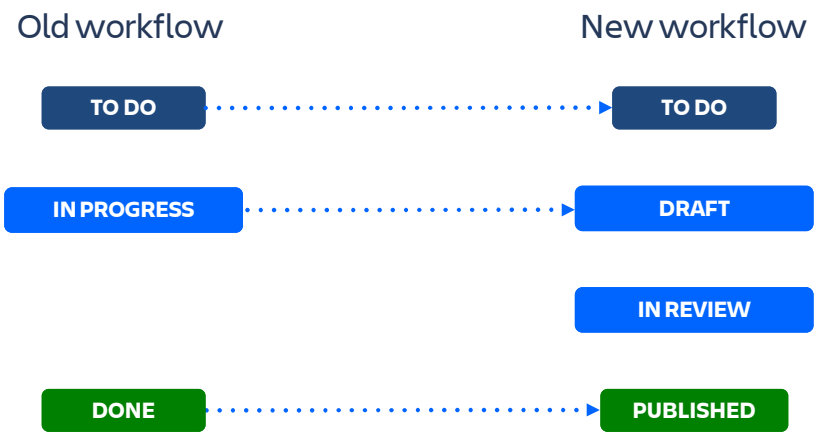
Try and keep your workflows as simple as possible. Aim for simple and scalable. Keep the number of statuses and transitions to a minimum to make them easy to use. Too many statuses and thus too many options of available workflow transitions will confuse the user.

If you need a new workflow, copy and customize an existing workflow rather than creating one from scratch. That way a lot of the work is already done for you.

Use transition screens sparingly, only when you need to get required information from the user, for example, a resolution.

It's risky to edit an active workflow that's in production. Once you publish it, the changes are live and if you made a mistake the consequences could be bad. Instead, copy the workflow, version it, ensure that it's name matches your artifact naming convention, make a note of what changed, then make your changes and save. Replace the current version with the new one after testing.

Migrating Issues



If you update a project's workflow scheme and change the workflow for all or some of the issue types, and there are inflight issues, the issues will need to be migrated. This is the process of mapping issues from statuses in the old workflow to statuses in the new workflow. You will be prompted to select which status in the new workflow issues should be mapped to. In this example, the old workflow has three statuses - TO DO, IN PROGRESS, and DONE. The new workflow has four statuses – TO DO, DRAFT, IN REVIEW, AND PUBLISHED. As there are more statuses and they are different names, you need to tell Jira which statuses to map. TO DO is mapped to TO DO and DONE is mapped to PUBLISHED. However for IN PROGRESS there's two possibilities to map to. You can only choose one so, in this case, we select DRAFT. You select this for each issue type that will use the new workflow in your project.

If there are a lot of issues to migrate (e.g. in the thousands), it could take some time. Also, once this process begins, it cannot be paused or canceled. Avoid editing or transitioning issues within your project while this process is taking place.

Before migrating a project to a new workflow, and if you're using Data Center, it's a good idea to run the database integrity checker. See https://confluence.atlassian.com/adminjiraserver/using-the-database-integrity-checker-938847667.html

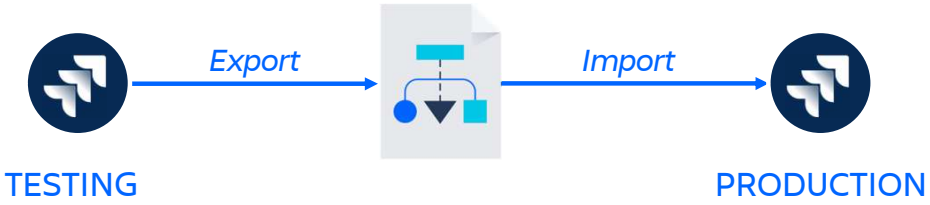
See how it's done

- Migrating issues

Instructor demo

Sharing Workflows

Export/import your workflows as **XML** or **Workflow**



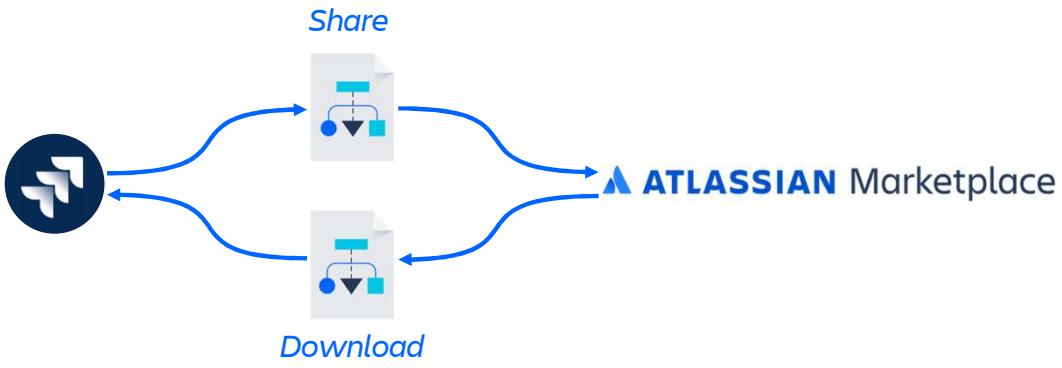
Using workflow export and import you can share your team's workflow with other teams in your organization on different Jira instances. This feature allows you to easily share and use workflows that other people have published, or to move a workflow from testing (or staging) to production in your own organization. Note, Jira administrators can export workflows but only Jira System administrators can import a workflow from a local Server instance.
For more information, see https://confluence.atlassian.com/adminjiraserver/sharing-your-workflow-938847433.html and https://confluence.atlassian.com/adminjiraserver/using-xml-to-create-a-workflow-938847525.html.
On Jira Cloud, only site administrators can export workflows and only workflows from the Atlassian Marketplace can be imported. More on that soon.
For more information, see https://confluence.atlassian.com/display/AdminJIRACloud/Importing+and+exporting+issue+workflows.

Importing & Exporting Gotchas

- Workflow app functionality may not be exported/imported
- Enable any disabled custom fields in the imported workflow before importing
- Importing XML workflows is supported for Server only

If you use apps that add functionality in workflows, aspects of the workflow might not be exported/imported. This is especially true when exporting 'as workflow'. If you export/import as XML the app functionality generally works as long as you have the app installed on the target Jira instance.
If the workflow that you are importing contains custom fields that are disabled, the workflow importer will not create these fields unless they are enabled before importing. You will receive a warning about this. To fix this, you need to enable the missing custom fields before proceeding with the import. For more information on custom fields in workflow imports, Sharing Your Workflow https://confluence.atlassian.com/adminjiraserver/sharing-your-workflow-938847433.html.
Importing XML workflows into Jira is supported for Jira Server only (not Jira Cloud).

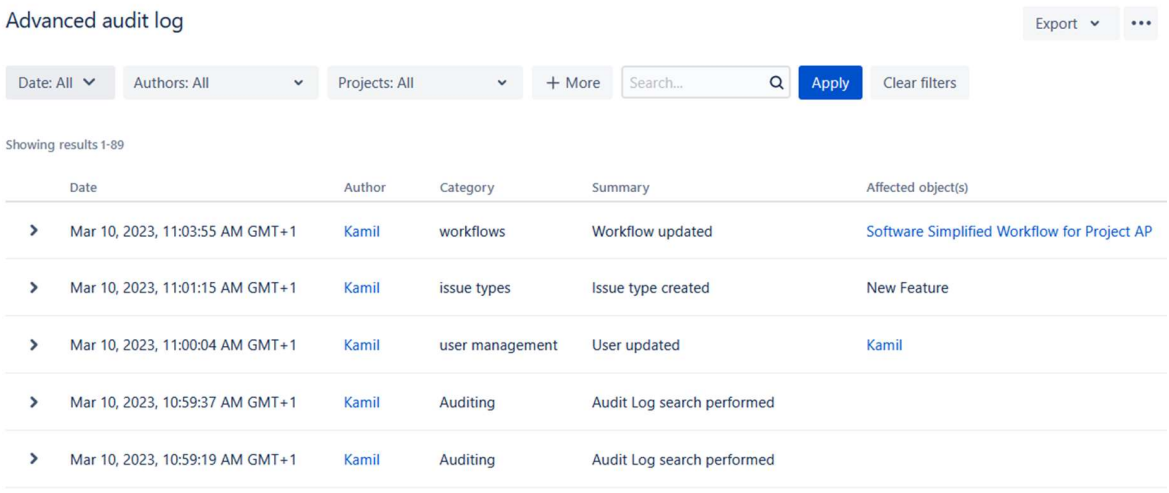
Sharing/Downloading Workflows on the Atlassian Marketplace



Using workflow export and import you can share your team's workflow with external parties in other organizations via the Atlassian Marketplace. You can also import workflows that others have created from the Atlassian Marketplace.
Only Jira administrators can export and import workflows from the Atlassian Marketplace.

Viewing the Audit Log

See who did what and when they did it



View the audit log to see what changes were made to workflows and who made them. This is useful information if you ever need to roll back a project's settings.
In the audit log you can see what workflows and workflow schemes were created, updated, deleted, copied, and renamed who did so, and when they did it. It also shows who added a workflow scheme to which project or removed a workflow scheme from a project.

Are you getting it?

Which of these methods can you use to get new workflows:

a. Import an XML workflow export from another Jira instance
b. Import a workflow export from another Jira instance
c. Extract the workflow from a system backup of another Jira instance
d. Download from the Atlassian Marketplace

Which of these methods can you use to get new workflows?

a. Import an XML workflow export from another Jira instance
b. Import a workflow export from another Jira instance
c. Extract the workflow from a system backup of another Jira instance
d. Download from the Atlassian Marketplace

Did you get it?

Which of these methods can you use to get new workflows:

- ✓ a. Import an XML workflow export from another Jira instance
- ✓ b. Import a workflow export from another Jira instance
- c. Extract the workflow from a system backup of another Jira instance
- ✓ d. Download from the Atlassian Marketplace

Answer: a, b, and d

To get new workflows into your Jira instance you can import a workflow export (as either XML or as workflow) from another Jira instance. This method is useful to share workflows among your team or to move workflows from a testing instance into production. You can also download new workflows from the Atlassian Marketplace.

You cannot extract a single workflow or a number of workflows from a system backup of another Jira instance.

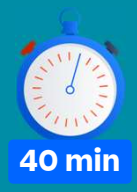
Takeaways

- Identify your key requirements and translate into Jira, before implementing
- Copy and customize an existing workflow rather than creating one from scratch
- Keep your workflows simple
- Test, test, test!

Don’t jump into implementing a new workflow in Jira. Spend time identifying the key requirements, breaking those down, and translating them into Jira before your actually start implementing. Measure twice, cut once!
If you need a new workflow, copy and customize an existing workflow rather than creating one from scratch. That way a lot of the work is already done for you.
Try and keep your workflows as simple as possible. Aim for simple and scalable. Keep the number of statuses and transitions to a minimum to make them easy to use. Too many statuses and thus too many options of available workflow transitions will confuse the user.
Test workflows in a test environment before going live to ensure your workflows behave as expected.

Lab 7 – Taking It to the Next Level

- Exercise 1
  - Add approval to a workflow
- Exercise 2
  - Add rejection to a workflow
- Optional Exercise 3
  - Share a workflow



Atlassian skills are in high demand in the job marketplace. The path to Atlassian certification is designed to help you deepen and expand your Atlassian skills and prepare you to take on your next challenge.

Follow the path to Certification – you'll propel your career, boost your skills and learn new ways to help your teams do their best work.

To find out more, visit **atlassian.com/certification**.

Training for Jira is a new Marketplace app containing a growing collection of short, interactive tutorials, designed to get teams over the hump and using Jira quickly.

Why Training for Jira?

- Help teams help themselves to training and reduce the number of Jira questions.
- Training for Jira scales – it's built right into Jira and is available to everyone.
- The tutorials are short and topic-focused; users can choose what to learn and when.
- The app also keeps track of users' progress so they can see where they left off, and so managers can track the progress of their teams.

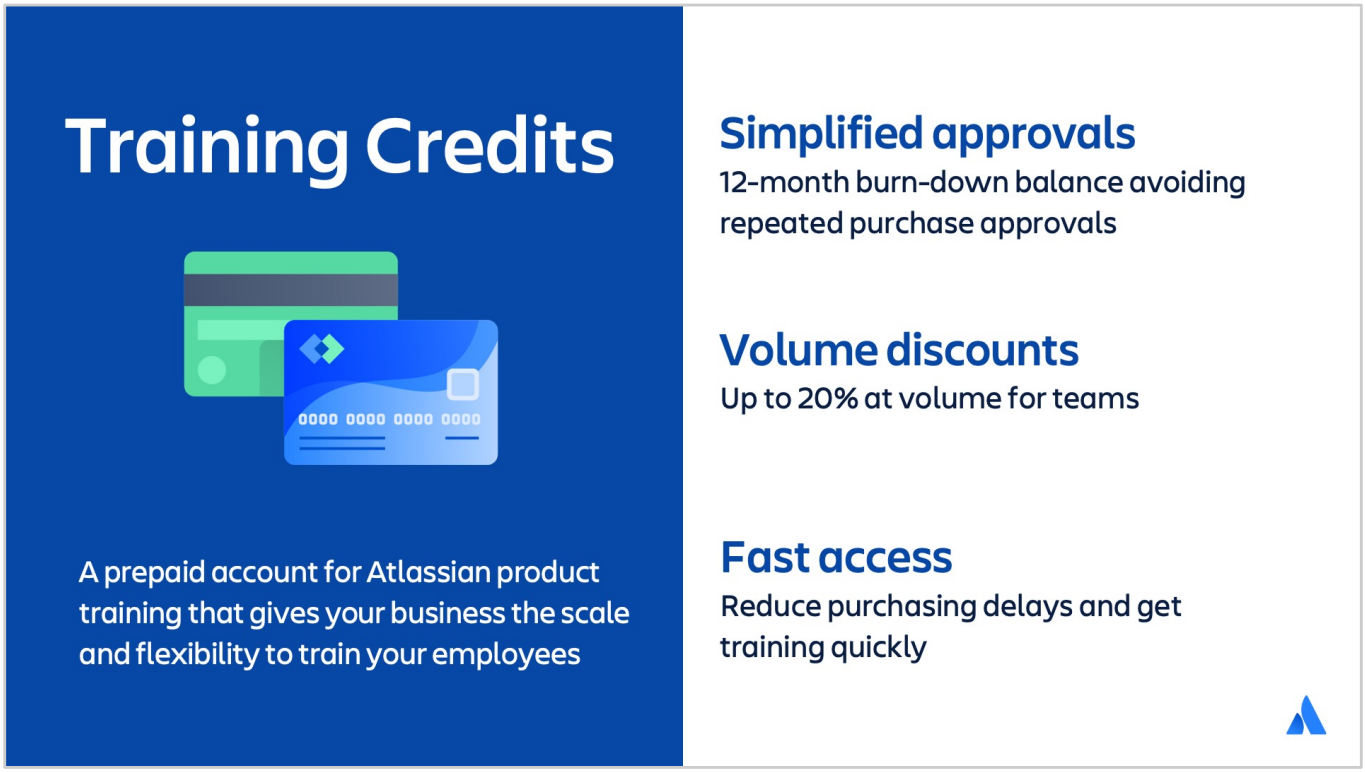
The Tutorials...

- Cover popular topics and tasks
- Introduce essential concepts
- Showcase in-product demos
- Include non-graded quizzes so you can see if you're getting it
- Have optional voice-over narration and closed captions

Available for Cloud, Server, and Data Center

Learn more

- Search for "Training for Jira" from marketplace.atlassian.com to download a free trial.
- Requires internet connectivity to access content
- Note: If you're using a firewall or have network restrictions where Jira is installed, you may need to whitelist IP addresses to access this app.



Simplified Approvals

Training credits allow your procurement team to establish a 12-month burn-down balance for Atlassian training. With this simple arrangement, you'll no longer need to process repeated purchase approvals for small transactions.

Volume Discounts

By pre-paying for training as an annual credits program, you can take advantage of volume training discounts to get a more effective return on your training budget.

Fast Access

With training credits, you reduce the purchasing delays for your business users and make it faster to access the training they need.

If you want to learn more, here are some additional resources to help.

Join your local Atlassian Community Events (ACE) group

What happens at an ACE?

A diverse group of Atlassian users, both technical and non-technical, come together to share best practices and enjoy food and drink.

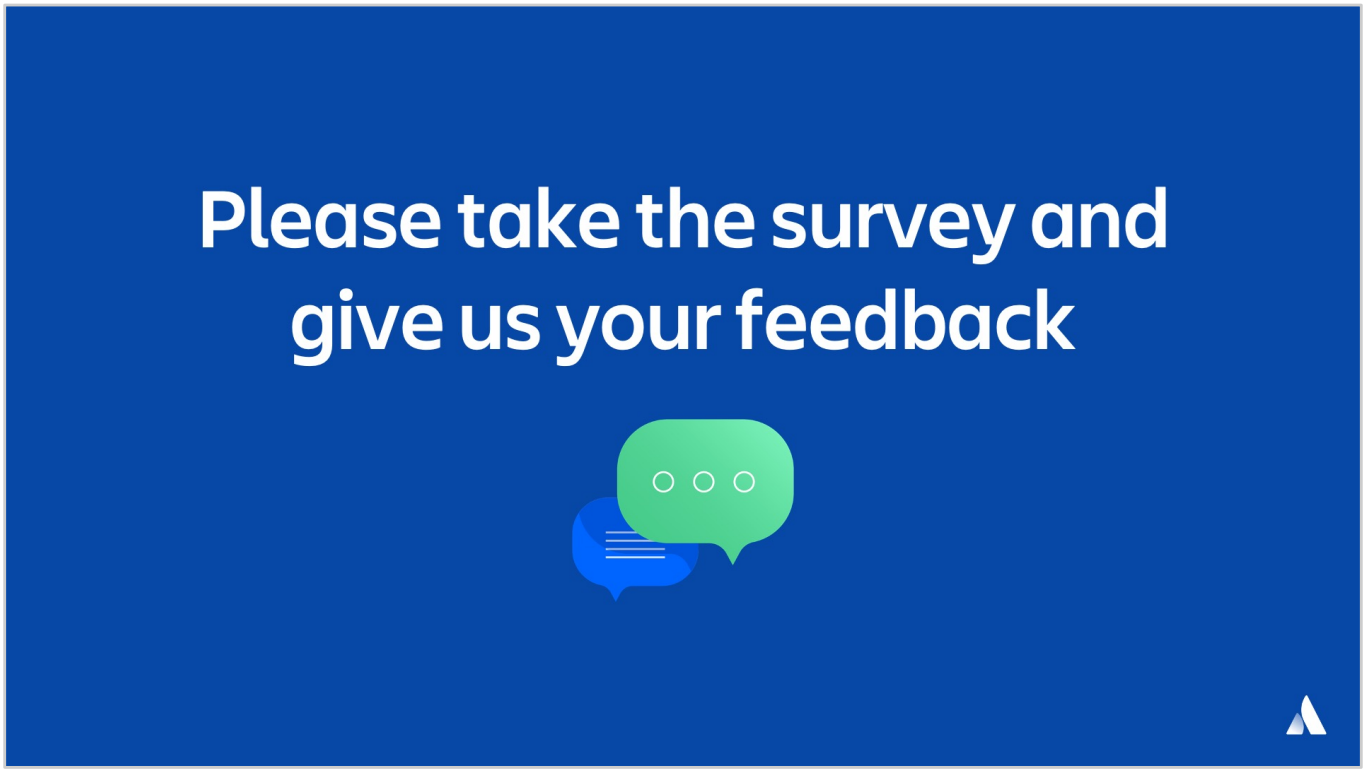
How do I join?

ACEs are in over 30 countries around the world. Go to ace.atlassian.com to join a group near you.

No ACE in your city?

You can start one! Go to ace.atlassian.com/leaders to sign up. You can also connect with fellow users on community.atlassian.com

We encourage you to join your local Atlassian Community Events (ACE) group!
An ACE is a diverse group of Atlassian users, both technical and non-technical, come together to share best practices and enjoy food and drink.
ACEs are in over 30 countries around the world. Go to ace.atlassian.com to join a group near you.
If there's no ACE in your city, you can start one! Go to ace.atlassian.com/leaders to sign up. You can also connect with fellow users on community.atlassian.com.



Now that you've finished the course, please take the survey and give us your feedback.

Congratulations on completing the course!