

Hands-On Agile Software Development with JIRA

Design and manage software projects using the Agile methodology



Packt

www.packt.com

By David Harned

Copyright 2018. Packt Publishing. All rights reserved. May not be reproduced in any form without permission from the publisher, except fair uses permitted under U.S. or applicable copyright law.

Hands-On Agile Software Development with JIRA

Design and manage software projects using the Agile methodology

David Harned



BIRMINGHAM - MUMBAI

Hands-On Agile Software Development with JIRA

Copyright © 2018 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Pavan Ramchandani
Acquisition Editor: Shriram Shekhar
Content Development Editor: Manjusha Mantri
Technical Editor: Abhishek Sharma
Copy Editor: Safis Editing
Project Coordinator: Prajakta Naik
Proofreader: Safis Editing
Indexer: Aishwarya Gangawane
Graphics: Jisha Chirayil
Production Coordinator: Aparna Bhagat

First published: July 2018

Production reference: 1280718

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78953-213-5

www.packtpub.com

*This book is dedicated to my wife, Stephanie, and to my children, Lily and Parker.
They help me manage our "project: family" every day. It is the most important,
challenging, and rewarding experience.*



mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributor

About the author

David Harned is a PMO director for Monotype and is a motivated and inspired leader of Agile thinkers. He is a design, usability, and customer experience advocate. David is an Agile believer and uses Scrum, Kanban, and Lean as well as hybrid approaches for project management. He holds many certifications in the project management and Agile domains, including PMI Project Management Professional, Digital Project Manager, Certified ScrumMaster, and more.

I'd like to thank my former manager, Chris Roberts, without whom I would never have stepped into this field. He made it possible for me to get where I am through his support. Also, I'd like to thank all the other Agile thinkers I have worked with in my teams. They have enriched my thought patterns while making sure I never forgot how to be humble.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface	1
Chapter 1: Get Started with Creating Your Project	5
Introduction to JIRA	5
What is JIRA?	5
How JIRA uses projects as a way to keep work organized	6
Creating an account with Atlassian	10
Project creation and management	14
Workflows	18
How to set up a project using scheme, screens, workflows, and permissions	21
Screens	22
Workflows	24
Permissions	27
Notifications	28
Summary	29
Chapter 2: Managing Work Items	30
Introduction to epics, stories, bugs, and tasks	31
Creating epics, stories, bugs, and tasks	32
Issue type attributes and adding and removing them	40
Managing items in the backlog	48
Creating and configuring our board	56
Summary	66
Chapter 3: Running Your Project in JIRA	67
Creating and starting a Sprint	67
The daily Scrum	75
Smaller stories or tasks	81
Closing the Sprint—the Sprint report	85
Summary	88
Chapter 4: Working with Reports	89
Versions and releases	89
Burndown report	95
Burndown example 1	95
Burndown example 2	96
Burndown report	97
Sprint report	98
Velocity chart	100

Velocity chart example	100
Release and epic burndowns	101
Release burndown example	101
Version and epic reports	103
Version report example	104
Summary	105
Chapter 5: Searching and Filtering on Issues	106
Issue searching using JQL	106
Simple and advanced JQL editors in JIRA	107
Saving and managing filters	110
Executing bulk changes	113
Creating new boards from saved filters	118
Summary	123
Chapter 6: Dashboards and Widgets	124
Creating and managing a dashboard	124
Adding gadgets to our dashboard	127
Sharing the dashboard	137
Summary	138
Other Books You May Enjoy	139
Index	142

Preface

JIRA software is an Agile project management tool that supports any Agile methodology. From Agile boards to reports, you can plan, track, and manage all your Agile software development projects with a single tool. With this book, you will learn to explore critical Agile terminologies and concepts in the context of JIRA Software.

Who this book is for

If you want to get started with JIRA software and learn how to run your JIRA projects the Agile way, this is the perfect book for you.

What this book covers

Chapter 1, *Get Started with Creating Your Project*, is about getting started with creating your project. We'll talk about how projects help you to keep your work organized and learn why JIRA is so popular and where it came from. We'll also talk about creating an account with Atlassian so that you can get started using JIRA on the cloud. We'll look into the projects themselves and how we use a project to organize all work items. We'll then look into the screens, workflows, and permissions, which will allow us to customize our project, and the views, notifications, and permissions that go along with it.

Chapter 2, *Managing Work Items*, is all about the difference between epics, stories, bugs, and tasks, learning what the different types of issues are, and why we would use each one. We'll talk about the attributes for those issues, learning what these different work item attributes are and how to customize them to fit your needs. We'll also cover managing items in the work, in the backlog, so that all that stuff will be a backlog in JIRA, and we'll talk about how to define, prioritize, and refine it. Then, we'll talk about creating and configuring a board, and how you would do that. If you're familiar with Scrum, you know what I mean by a Scrum board.

Chapter 3, *Running Your Project in JIRA*, is about running the project. We'll create and start a Sprint. We'll use our backlog to refine the work and then plan and begin the Sprint iteration. We'll also look at the daily Scrum and how we use JIRA to keep the team aligned, and how to know whether we're on track to meet our commitments. We'll focus on the differences between smaller stories or tasks, and when do we use each one, and then we'll talk about how to close a Sprint, and learn how to end that Sprint, and what to do with any work that hasn't been completed.

Chapter 4, *Working with Reports*, explores all about versions and releases—what they are and how they're different from each other. We'll talk about burndowns, about Sprint reports, and how to read those to determine whether or not your team is doing well. We'll also take a look at velocity charts, which we can use to determine the performance of the team. We will take a look at releasing epic burndowns, as well as versions and epic reports, which give you the ability to do forecasting, which is very powerful.

Chapter 5, *Searching and Filtering on Issues*, is about JQL, what it is, how to write queries in JIRA using simple and advanced editors, and how to export your results. We'll talk about saving and managing filters, and then executing bulk changes, and then how to use those filters to create new boards. This will give you new views of your work items in JIRA.

Chapter 6, *Dashboards and Widgets*, teaches us what a dashboard is, how you would use it, the different things you can put on a dashboard, the different layouts you can have for it, and then how to share it so that you can ensure that you're able to broadcast the results of the team and how things are going. Hence, there aren't a lot of prerequisites for this course, just a couple of things that I thought would be helpful—one is to have a basic knowledge of Scrum. We'll reference Scrum a fair amount as we're running an Agile project in JIRA, so that would be helpful for you to know. Still, it would be nice to have at least one team of people that are looking to work together, because that's what JIRA is really great for, having a team of people work together, not necessarily just one person working on something. Although it can work for that, having a team that you can apply these concepts to once you've learned them will be really helpful.

To get the most out of this book

- You will need to be familiar with the basics of JIRA, from both the end user and administrator perspectives
- Experience with workflows, custom fields, and other administrative JIRA functions will be useful

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: https://www.packtpub.com/sites/default/files/downloads/HandsOnAgileSoftwareDevelopmentwithJIRA_ColorImages.pdf.

Conventions used

There are a number of text conventions used throughout this book.

CodeInText: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Let's create another Sprint. We'll call this one `FP1 Sprint 1` and include `This is my first Sprint` as the Sprint goal."

Bold: Indicates a new term, an important word, or words that you see on screen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "If we go back to our **Backlog** in the upper right corner, we can see that we have our **Board settings**, so we'll click that, and then, under our **SETTINGS**, we've got **Estimation**."



Warnings or important notes appear like this.



Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email feedback@packtpub.com and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at questions@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packtpub.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packtpub.com.

1

Get Started with Creating Your Project

In this chapter, we'll take a look at projects that help to keep our work organized. We'll learn in detail about JIRA and how it can be used in managing all our projects. Let's get started.

In this chapter, we will learn about the following topics:

- Introduction to JIRA
- Creating an account with Atlassian
- Project creation and management
- How to set up a project using scheme, screens, workflows, and permissions

Introduction to JIRA

This section is on JIRA software essentials. In this section, we will learn about what JIRA is, and getting started with creating projects and how JIRA can organize the work that we include within it.

What is JIRA?

JIRA has been in place for a while, and it originally as an issue ticketing system, with a bug tracking kind of software, but, as project management has evolved over the years, Agile processes have become increasingly popular. JIRA has become a very effective Agile management tool for both Scrum and for common, and is now primarily used in that way.

JIRA has three different packages that it offers at this time:

- JIRA Core
- JIRA Software
- JIRA Service Desk

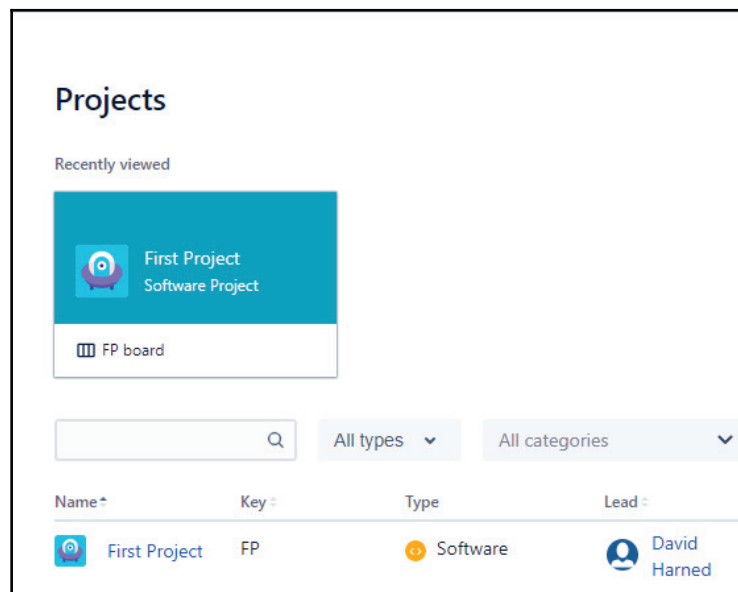
These packages include the base software of JIRA Core and also include Agile project management features.

JIRA has been used extensively and it has a huge community with lots of add-ons, which allow for planning, tracking, releasing, and reporting. Here's a link we can use to learn more about JIRA: <https://www.atlassian.com/software/jira>.

How JIRA uses projects as a way to keep work organized

Here are the following steps to view projects in JIRA:

1. Log in using the JIRA credentials. We will now notice that we've already created a **First Project**. It's a **Software** type project:



Project view

2. To understand how this was done, click on **Create project**.
3. Name this project `Second Project`. We can now see in the following screenshot that we have a **Scrum** template, and we could change that to something else if we wanted to, but for now, we'll leave it as it is and click on the **Create** button:

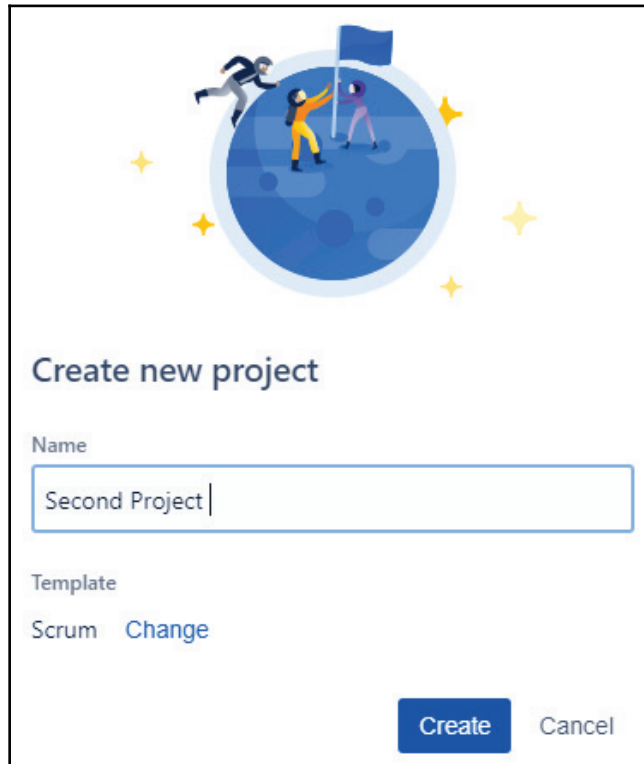


Illustration of three people climbing a globe with a flag at the top.

Create new project

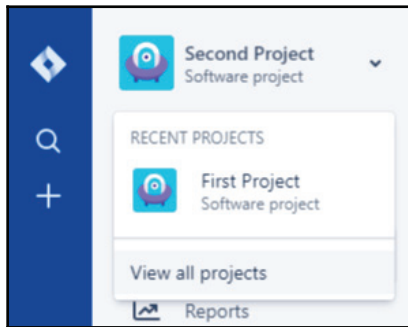
Name

Template

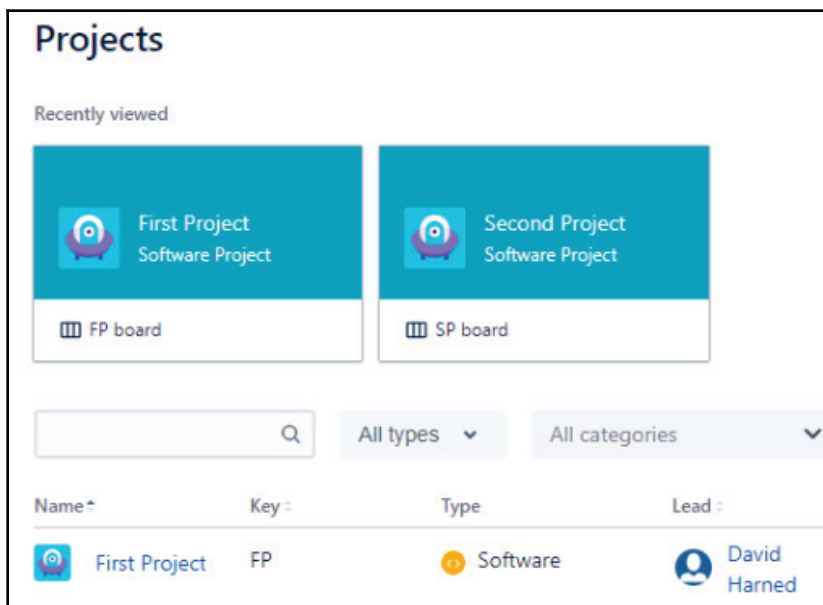
Scrum [Change](#)

Create Cancel

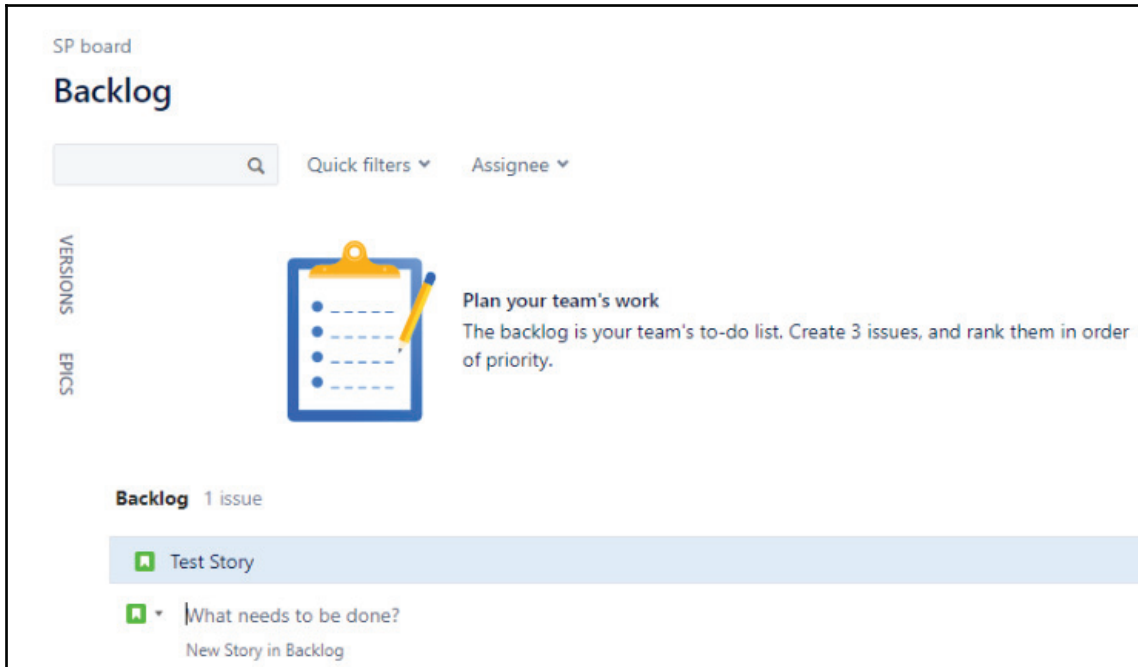
4. We have the **Second Project**, and, as we can see in the following screenshot, we've got our **First Project** in our **Second Project**:



5. Go to **View all projects**, where we can see all of the projects:



6. Click on **Second Project**. We can see in the following screenshot what this backlog view looks like. This is where we can create a test story and put items in our backlog:



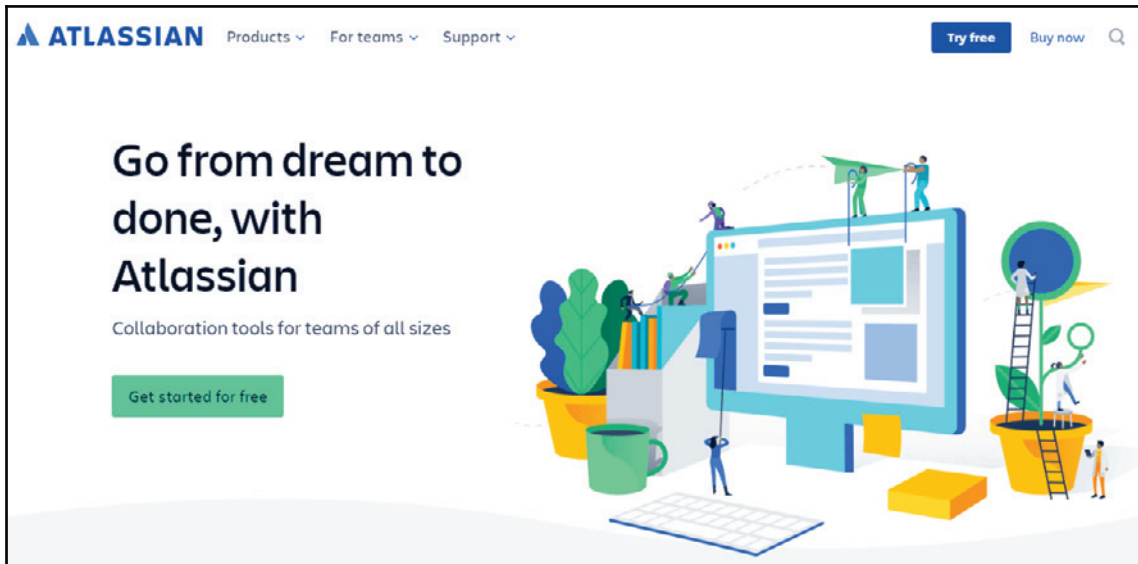
JIRA uses projects to help us organize our work and create a holding place for everything. It uses a key, which is a three or four digit ID, which we can reference as well.

We'll go more into detail about what all of these different things are in the UI, but for now, it's important to note that projects are what JIRA uses in order to organize our work.

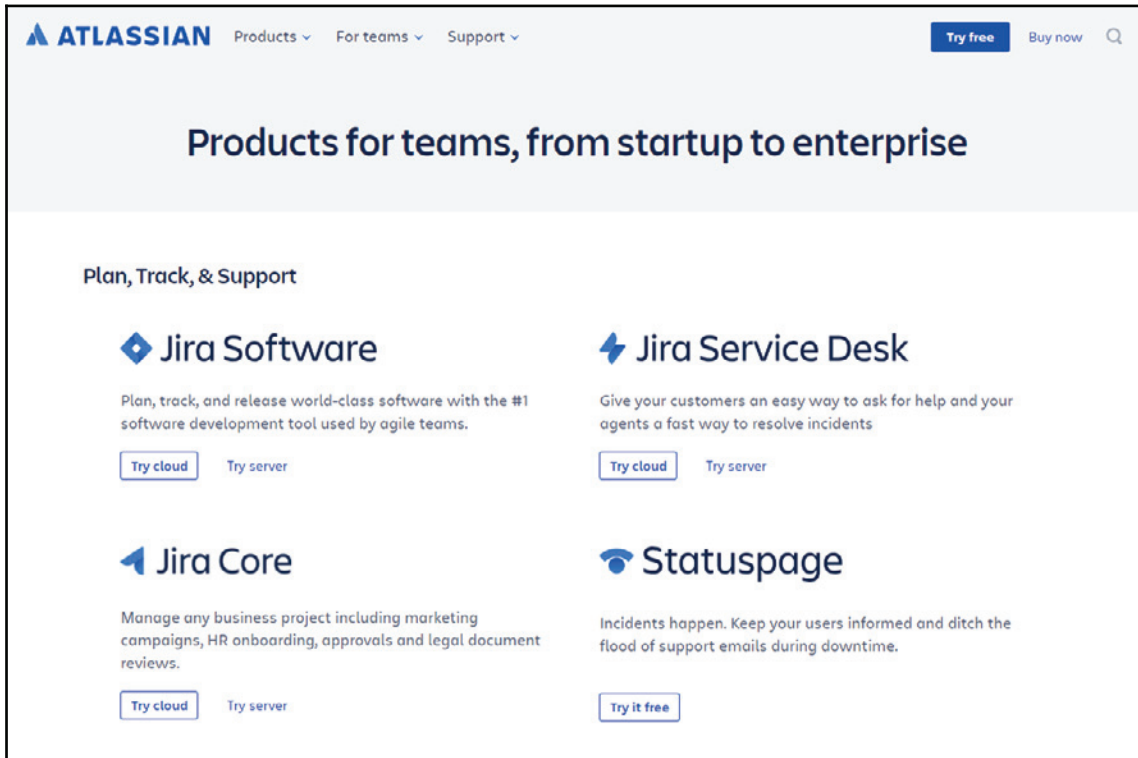
Creating an account with Atlassian

In this section, we're going to learn how to create an account and how to set up our JIRA software so that we can begin using it for project management:

1. Open the *Atlassian* website. We'll see that there's plenty of information about the company, about some of the different software products that they have, and more. We'll go to the **Try free** button at the top of the page:




2. We'll see that they've got some options. We're going to use the **Jira Software**. They have a server version and a cloud version:






The screenshot shows the Atlassian website's product page. At the top left is the Atlassian logo and navigation links for 'Products', 'For teams', and 'Support'. On the top right are 'Try free', 'Buy now', and a search icon. The main heading is 'Products for teams, from startup to enterprise'. Below this, a section titled 'Plan, Track, & Support' features four product cards: 'Jira Software' (described as the #1 software development tool), 'Jira Service Desk' (for customer support), 'Jira Core' (for business project management), and 'Statuspage' (for incident communication). Each card includes a 'Try cloud' button and a 'Try server' link.

3. We're going to be using the cloud version, so let's select that. We can see that we can try it for free for seven days, but we've also got some different options for once it becomes a paid subscription. Since we're just a small team, we're going to go ahead and take the first option, which is **\$10 a month**, and we're going to **Try it free**:

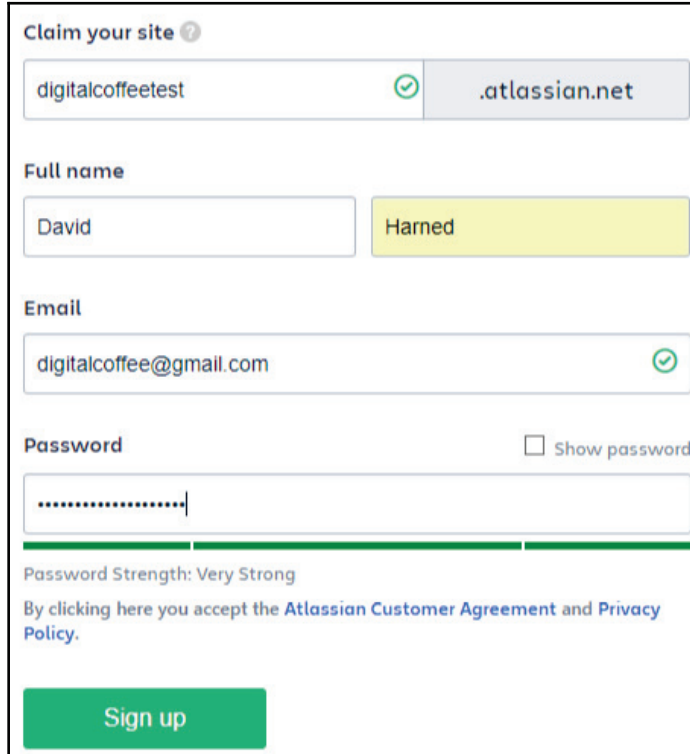


Try our products in the Cloud, free for 7 days

72% of our customers combine our products

<p>Jira Software</p> <p>From \$10 /month</p>  <p>Plan, track, and release world-class software</p> <p>Try it free</p> <p>Jira Software</p>	<p>Most Popular</p> <p>Jira Software + Documentation</p> <p>From \$20 /month</p>  <p>Centralize and share ideas to supercharge development</p> <p>Try it free</p> <p>Jira Software Confluence</p>	<p>Jira Software + Help Desk</p> <p>From \$20 /month</p>  <p>Create, build, and support software, from end to end</p> <p>Try it free</p> <p>Jira Software Jira Service Desk</p>
---	--	---

4. We'll see that on the following screen, it's going to ask us to go ahead and set up our URL. We'll choose `digitalcoffeetest`, we'll put in our name, our email, and we can see that we then have to add a password:



The screenshot shows a 'Claim your site' form with the following fields and elements:

- Claim your site** (with a help icon): A text input containing 'digitalcoffeetest' with a green checkmark, and a dropdown menu showing '.atlassian.net'.
- Full name**: Two text input fields. The first contains 'David' and the second contains 'Harned'.
- Email**: A text input containing 'digitalcoffee@gmail.com' with a green checkmark.
- Password**: A text input with masked characters (dots) and a 'Show password' checkbox.
- Password Strength**: A green progress bar and the text 'Password Strength: Very Strong'.
- Terms and Conditions**: A link that reads 'By clicking here you accept the [Atlassian Customer Agreement and Privacy Policy](#)'.
- Sign up**: A large green button at the bottom.

Once we do that, the website will go ahead and send us an email to validate the data that we have just given them. Once we check our email and validate that the information is correct, we will then go back to the site and log in.

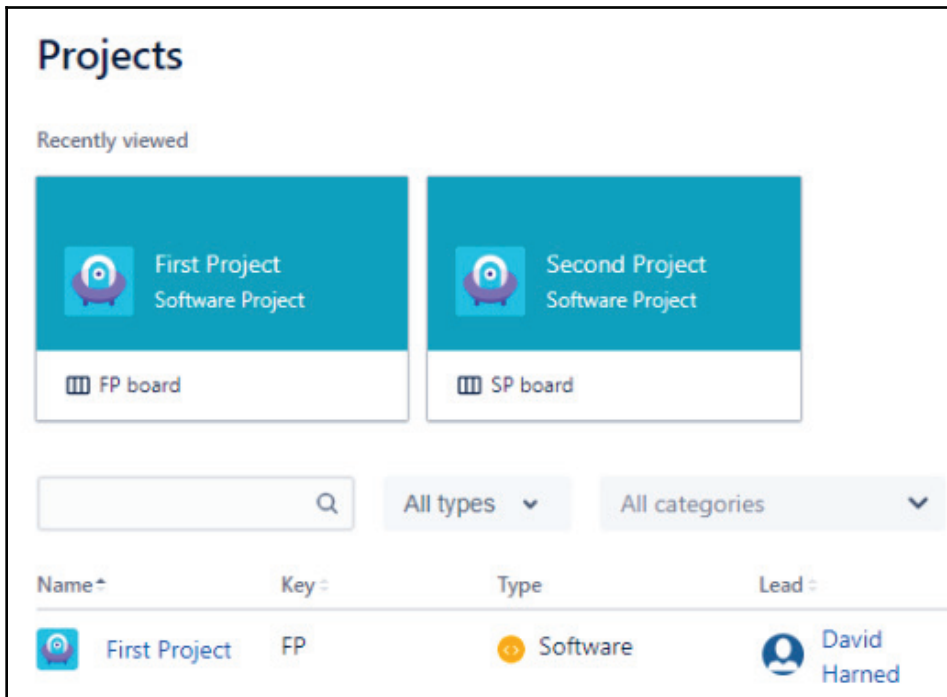
Creating an account is that easy.

Project creation and management

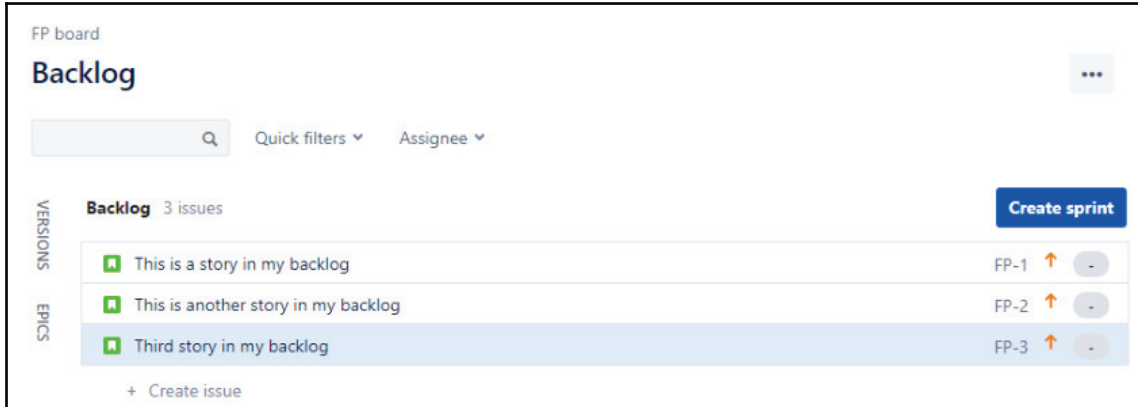
In this section, we're going to talk about creating projects and then managing those projects in JIRA.

The steps to create and manage projects in JIRA are as follows:

1. Select projects in the left-hand menu. We can see that we have a **First Project** and then a **Second Project**:



2. Select the **First Project**. This will bring us to our **Backlog** view of the project. The backlog will store all of the stories, bugs, and different issue types that we want inside this project:

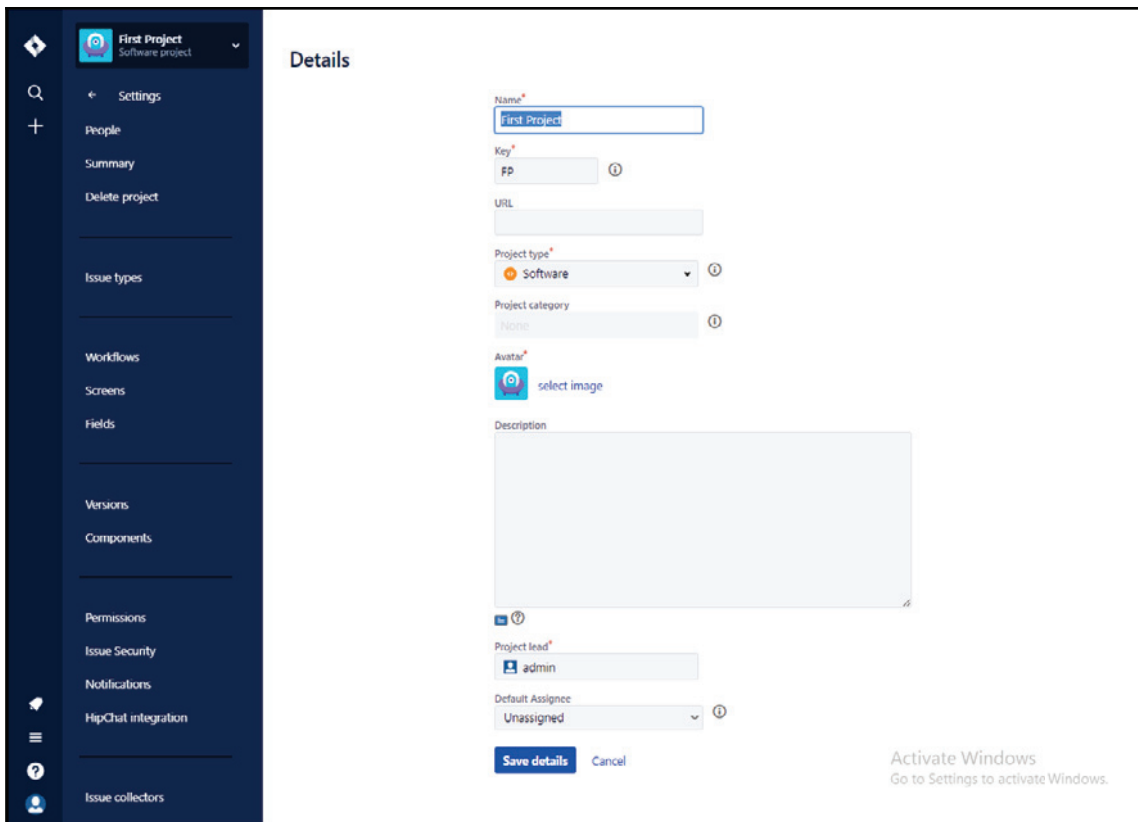


Backlog view of the project

On the left, we'll see that we have the following different options. First, we have a search option. In our search, we can either type or we can also hit the slash key (/) on our keyboard. We've got some options regarding filtering and looking at assignees:

- **Sprints:** We can select this and get the board view for the current Sprint
- **Versions:** This determines who gets report releases when we do a release of a version, in which case we can take a look at those issues, which will allow us to perform queries for different issues
- **Components:** These are a grouping of work items

3. Go down to **Settings**. This will take us to the following screen:



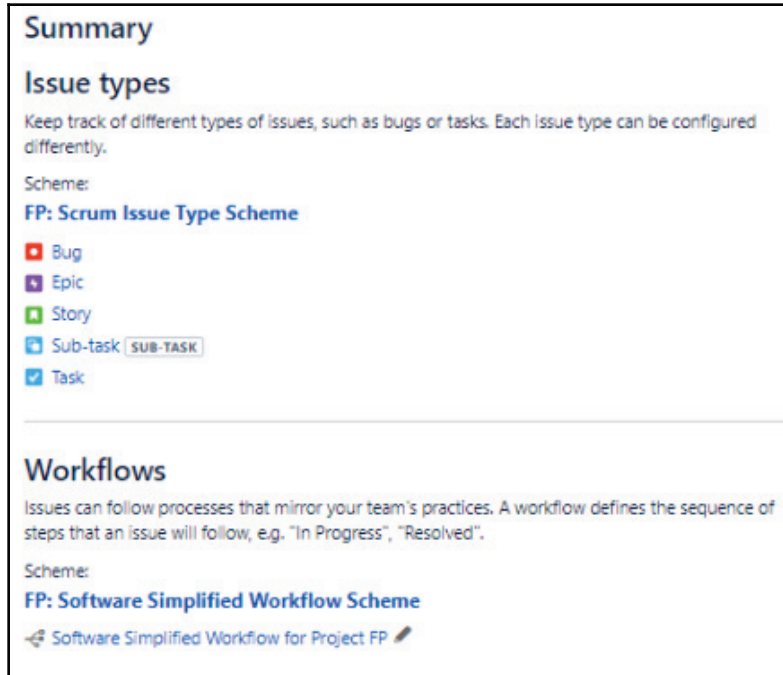
The screenshot shows the 'Details' page for a project named 'First Project'. The left sidebar contains navigation options: Settings, People, Summary, Delete project, Issue types, Workflows, Screens, Fields, Versions, Components, Permissions, Issue Security, Notifications, HipChat integration, and Issue collectors. The main content area is titled 'Details' and contains the following fields:

- Name***: Text input field containing 'First Project'.
- Key***: Text input field containing 'FP'.
- URL**: Empty text input field.
- Project type***: Dropdown menu with 'Software' selected.
- Project category**: Text input field containing 'None'.
- Avatar***: Image selection button labeled 'select image'.
- Description**: Large empty text area.
- Project lead***: Text input field containing 'admin'.
- Default Assignee**: Dropdown menu with 'Unassigned' selected.

At the bottom of the form are two buttons: 'Save details' and 'Cancel'. In the bottom right corner, there is a watermark: 'Activate Windows Go to Settings to activate Windows.'

As we can see, we've got our name, we've got a key, URL, the type of project that this is, and we're going to stick with the software that's going to let us perform Agile processes like Scrum. We can categorize, we can select an image and description, we can also decide who is the admin on this project, and we can decide, when we create a new item in the backlog, whether it gets assigned to the project lead, which, in this case, would be the admin or we can even leave it as unassigned.

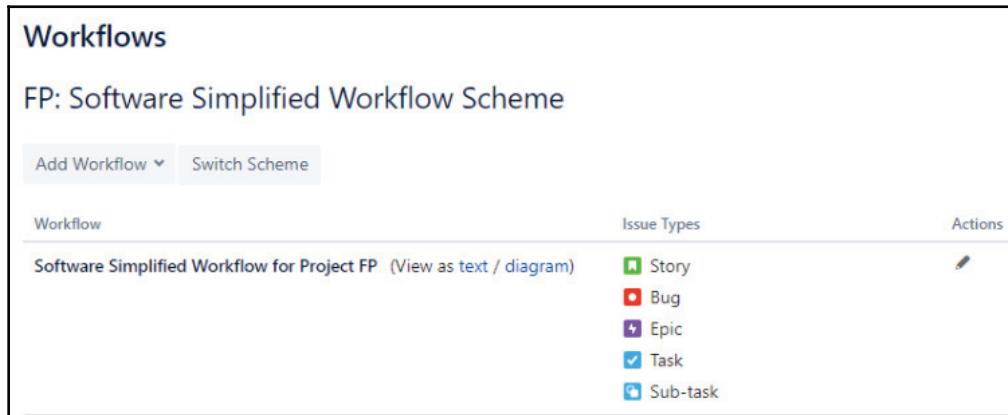
There are a lot of options on the left, but we're going to take a look at the summary, because the summary is going to give us a view of all of those different options, as shown in the following screenshot:



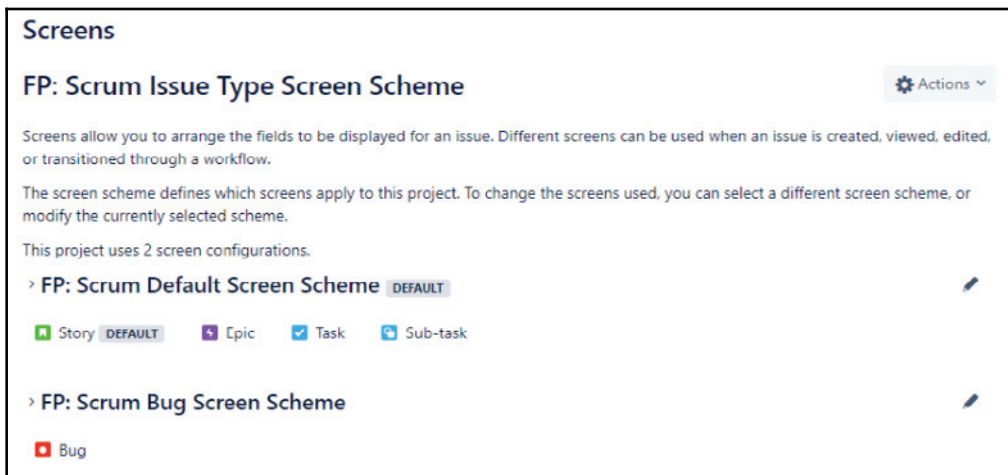
Summary

Workflows

First, let's take a look at workflows so that we can understand how workflows operate. We'll go into more detail in the next section about workflows, but what we really need to understand is that this will control the way that the issues move from **To Do**, to **In Progress**, to **Done**, and this allows us to customize the way that it works:



If we take a look at screenshots for this project, we can see that we're using a Scrum issue type, so this will allow us to select the items that appear on those different issue types, which are things such as story points, assignee, and acceptance criteria:



We will now look into the **Fields**. This allows us to control the fields that are available:

Fields

System Default Field Configuration ⚙️ Actions ▾

Different issues can have different information fields. A field configuration defines how fields behave for the project, e.g. required/optional; hidden/visible. The field configuration scheme defines which fields apply to this project. To change the fields used, you can select a different field configuration scheme, or modify the currently selected scheme.

This project uses only 1 field configuration.

▾ **Default Field Configuration** DEFAULT SHARED BY 2 PROJECTS ✎

These 5 issue types...

- Story DEFAULT
- Bug
- Epic
- Task
- Sub-task

...use this field configuration

Name	Required	Renderers	Screens
Affects Version/s		Autocomplete Renderer	3 screens
Approvers <small>Contains users needed for approval. This custom field was created by Jira Service Desk.</small>			No screens
Assignee			7 screens
Attachment			5 screens
Change completion date <small>Specify the completion time for the change request</small>			No screens

We can do things like setting up and creating components.

1. We will now click on **Components** on the left and we will call this component Test:

Create component

Name *

Description

Component lead

Default assignee

Save
Cancel

2. This will now allow us to assign this component to different issue types. We can do things such as set permissions, as can be seen in the following screenshot:

Default software scheme Permission helper ⚙️ Actions ▾

Project permissions allow you to control who can access your project, and what they can do, e.g. "Work on Issues". Access to individual issues is granted to people by issue permissions.

The permission scheme defines how the permissions are configured for this project. To change the permissions, you can select a different permission scheme, or modify the currently selected scheme.

[Learn more about project permission schemes](#)

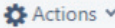
Project Permissions

Permission	Users / Groups / Project Roles
Administer Projects Ability to administer a project in Jira.	Project Role (Administrators) Project Role (atlassian-addons-project-access)
Browse Projects Ability to browse projects and the issues within them.	Application access (Any logged in user) Project Role (atlassian-addons-project-access)
Manage sprints Ability to manage sprints.	Application access (Any logged in user) Project Role (atlassian-addons-project-access)

And we can even set notifications as follows:

Notifications


Default Notification Scheme



SHARED BY 2 PROJECTS

Jira can notify the appropriate people of particular events in your project, e.g. "Issue Commented". You can choose specific people, groups, or roles to receive notifications.

The notification scheme defines how the notifications are configured for this project. To change the notifications, you can select a different notification scheme, or modify the currently selected scheme.

- Email: `jira@` [redacted] `.atlassian.net` 

Events	Notifications
Issue Created	All Watchers Current Assignee Reporter
Issue Updated	All Watchers Current Assignee Reporter
Issue Assigned	All Watchers Current Assignee Reporter

We can see that when an issue is created, we're going to notify all watchers, the current assignee, and the reporter when the issues are updated, and more besides. We can also customize this. If you're one of those people who receives way too many emails already, you might want to slim this down a little bit so that we're really only being notified about the most important actions. We know now how to create and manage our projects in JIRA.

How to set up a project using scheme, screens, workflows, and permissions

In the last section, we learned about the configuration scheme of JIRA, and how to set up projects with regard to screens, workflows, permissions, and even notifications.

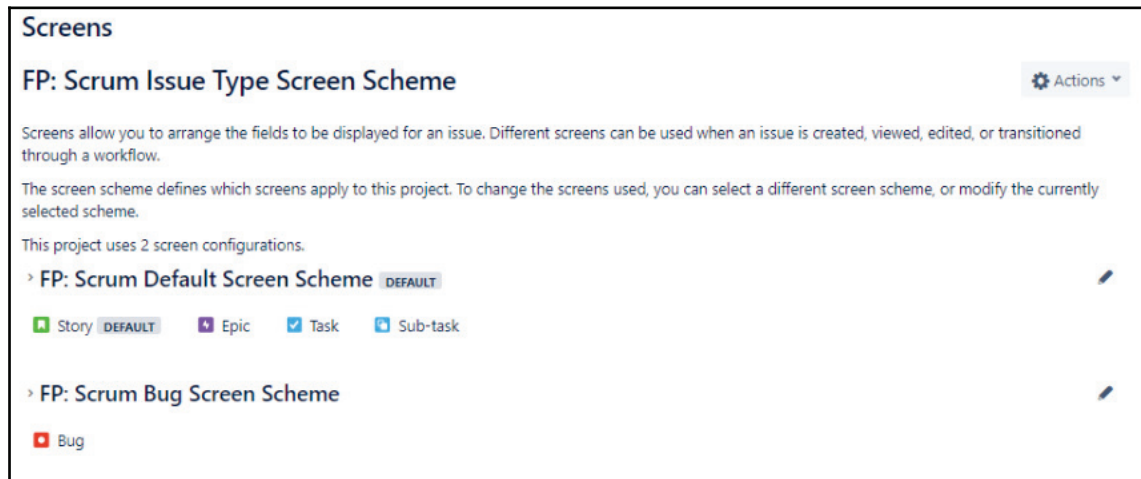
In this section, we're going to learn about the following in detail:

- Screens
- Workflows
- Permissions
- Notifications

Screens

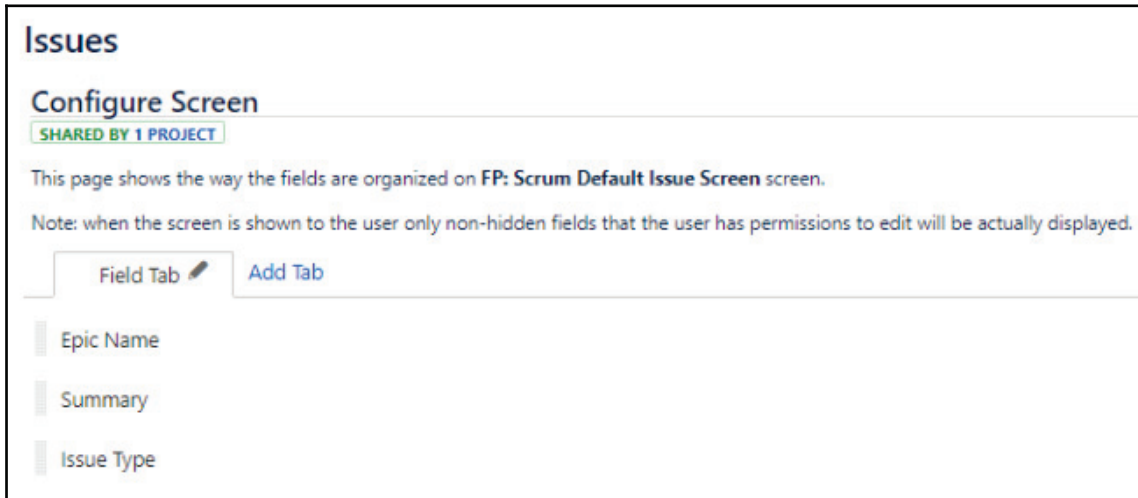
We are going to flip over to the JIRA account and look at our project view:

1. First, we will select a project, and once we have our project loaded up, we're going to go ahead and choose our settings for this project. We'll go over to the left-hand menu and select **Settings**. As we learned previously, we will set up our settings.
2. We saw a little bit about this in the previous section, but we're going to learn more deeply about **Screens**. We can see that we use what's called a Scrum issue type screen scheme. We've got a default screen scheme that will cover the different issue type **Story**, **Epic**, **Task**, and **Sub-task**, and then we've got a bug screen scheme that will cover the issue type **Bug**, as shown in the following screenshot:



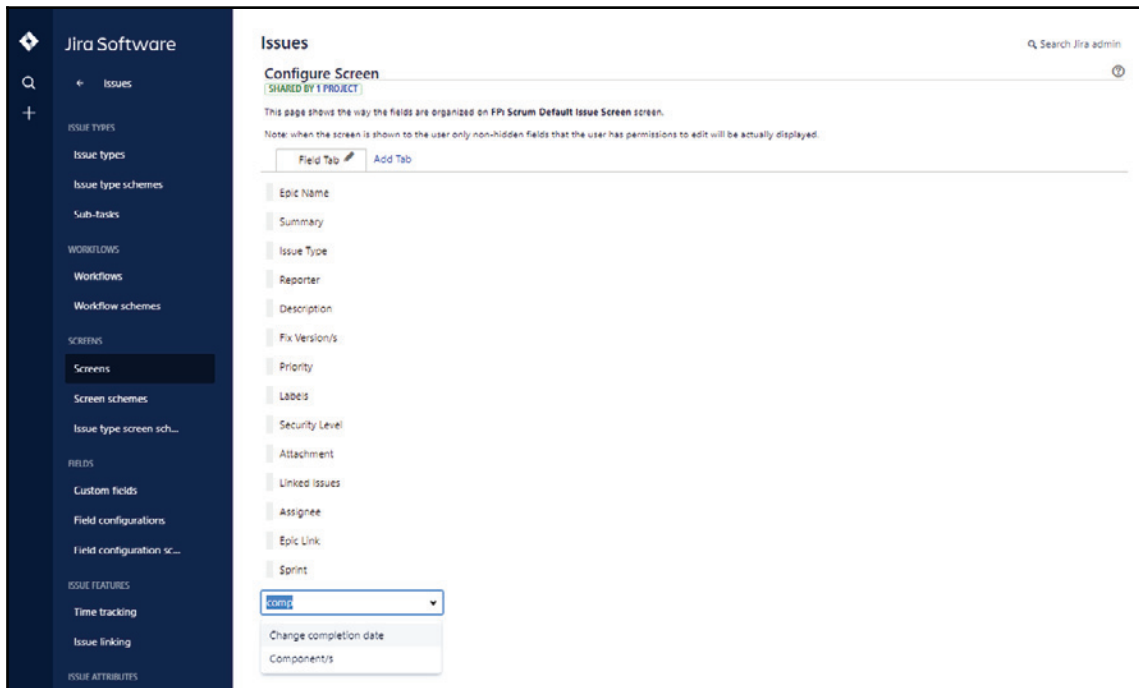
The screenshot shows the 'Screens' configuration page in JIRA. The title is 'FP: Scrum Issue Type Screen Scheme'. Below the title, there is a description: 'Screens allow you to arrange the fields to be displayed for an issue. Different screens can be used when an issue is created, viewed, edited, or transitioned through a workflow.' Another line of text states: 'The screen scheme defines which screens apply to this project. To change the screens used, you can select a different screen scheme, or modify the currently selected scheme.' Below this, it says 'This project uses 2 screen configurations.' There are two screen configurations listed: 'FP: Scrum Default Screen Scheme' (marked as 'DEFAULT') and 'FP: Scrum Bug Screen Scheme'. Under the 'FP: Scrum Default Screen Scheme', there are four issue types listed: 'Story' (DEFAULT), 'Epic', 'Task', and 'Sub-task'. Under the 'FP: Scrum Bug Screen Scheme', there is one issue type listed: 'Bug'. There are edit icons (pencil) next to each screen configuration.

3. Click the edit icon on the right. We can then take a look at the default issue screen, as follows:



4. This will give us control over all of the different fields that appear in an issue type. There may be some that we don't need to use, because our organization doesn't use them or we don't find value in those, in which case removing them from the interface will make things faster, so we don't need to have a view of those. The way we would do that is by dragging and moving these into different orders, but let's say for instance that we didn't want components. Therefore, we can easily just remove those by clicking on **Remove**. The components will no longer appear on any of our issue types with the exception of bug, right? We haven't configured the bug one yet.

5. At the bottom, we can see the ability to select the field, so we can actually select a field if we want to add something back, like `Components`:



Or, we could even just type `components` and add it back that way, and we'll drag it back up to the right before the description where we had it before. That's how we use screens in JIRA. Next, let's take a look at workflows.

Workflows

From the options on the left-hand side, click on **Workflows**.

What the workflow is going to let us do is allow us to see the different ways that a status can affect another status, and how items can move from one status to the next so that we can see how this works:

1. Click on **FP: Software Simplified Workflow Scheme**, as shown in the following screenshot:

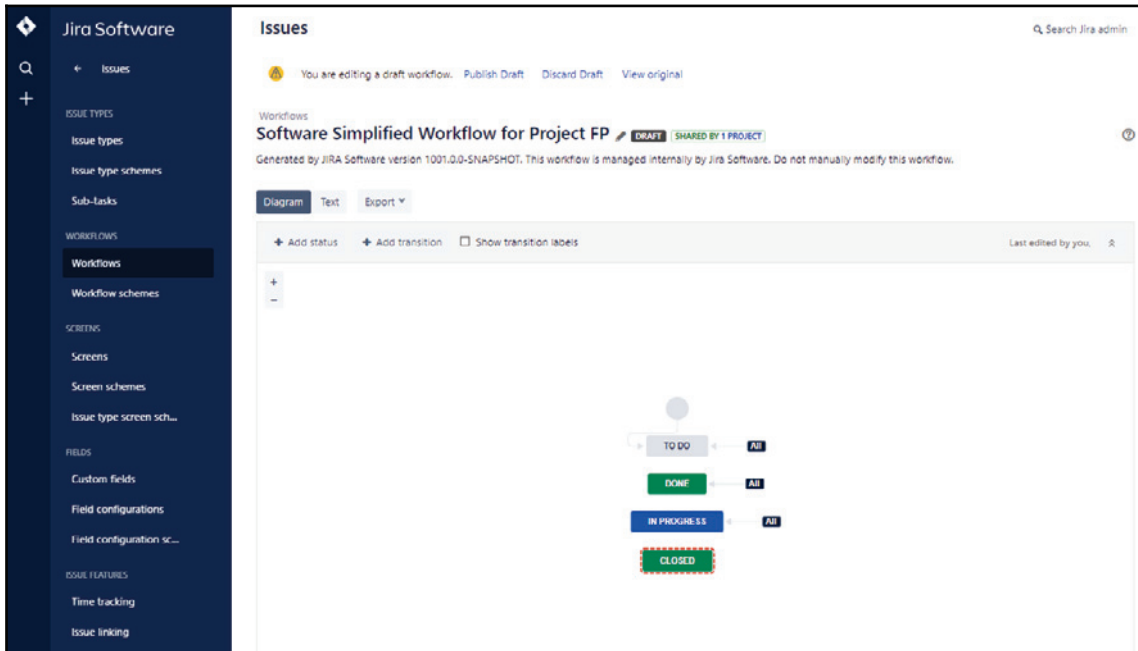


This will take us to the following screen:



2. Any thing, and any type, can be move to **TO DO**. Once we create any type, we can automatically move it **TO DO**, any type can moved to **DONE**, and any type can move to **IN PROGRESS**.
3. We can actually add a status as **Closed** by clicking on **Add status**.

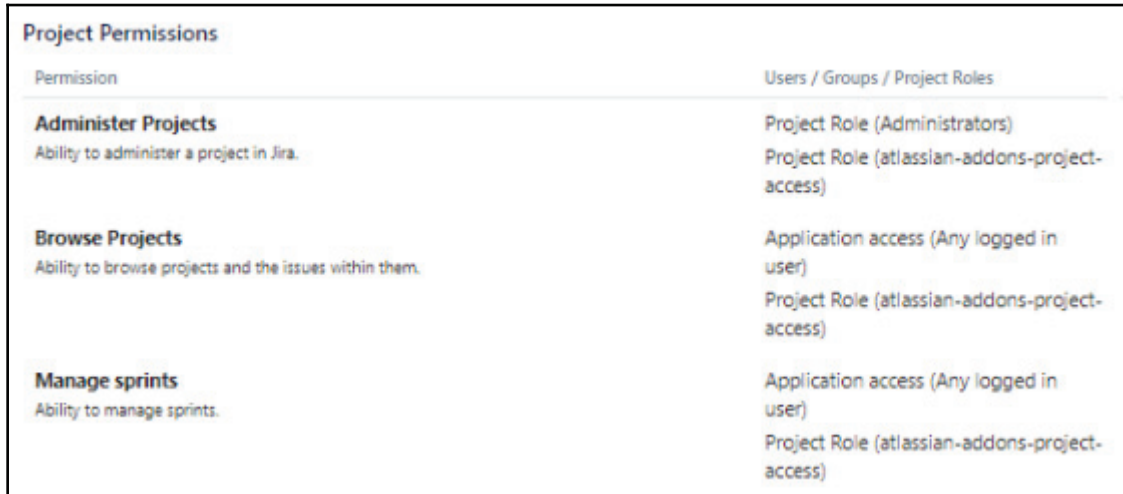
4. Drag the **Closed** status to the bottom, below **IN PROGRESS**, and now we have a new status:



Then, we can allow different transitions to happen to this closed status. One example that we might want to do would be, let's say, if we have story points assigned to an item, or perhaps hours assigned to an item. When we move that through the workflow, and that item goes from **IN PROGRESS** to **DONE**, when we move it to **CLOSED**, we may want to have the remaining hours update itself to zero. This is an example of something that we might want to do with that workflow.

Permissions

We want to take a look at permissions, so we'll bring up the **Permissions** options for our default software scheme, which is what we're using for our Scrum project, as can be seen in the following screenshot:



The screenshot shows the 'Project Permissions' configuration page in Jira. It features a table with two columns: 'Permission' and 'Users / Groups / Project Roles'. Three permissions are listed: 'Administer Projects', 'Browse Projects', and 'Manage sprints'. Each permission has a description and a list of associated users or roles.

Permission	Users / Groups / Project Roles
Administer Projects Ability to administer a project in Jira.	Project Role (Administrators) Project Role (atlassian-addons-project-access)
Browse Projects Ability to browse projects and the issues within them.	Application access (Any logged in user) Project Role (atlassian-addons-project-access)
Manage sprints Ability to manage sprints.	Application access (Any logged in user) Project Role (atlassian-addons-project-access)

Project Permissions

Since we're the only one that's actually accessing and controlling this, we have access to everything, but if we had more people that were assigned to this project, then we would be able to identify who can do what, both from a project permissions perspective as well as from an issue permissions perspective.

Notifications

We can see that we have a **Notifications** option under our **First Project**. We learned a little bit about this in the last section, but we will learn more about it now:

Notifications

Default Notification Scheme ⚙️ Actions ▾

SHARED BY 2 PROJECTS

Jira can notify the appropriate people of particular events in your project, e.g. "Issue Commented". You can choose specific people, groups, or roles to receive notifications.

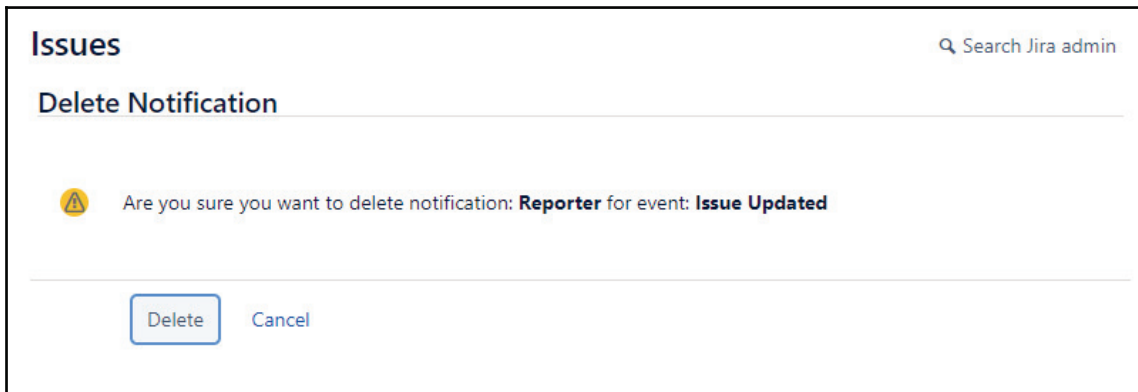
The notification scheme defines how the notifications are configured for this project. To change the notifications, you can select a different notification scheme, or modify the currently selected scheme.

- Email: jira@[redacted].atlassian.net ✎

Events	Notifications
Issue Created	All Watchers Current Assignee Reporter
Issue Updated	All Watchers Current Assignee Reporter

But let's say we were the person who reported an issue, and, when that issue was created, we would want the current assignee, the person that's assigned to that issue, to be notified, and we'd also want any of the watchers of that item to be notified as well. We, as the reporter, would be notified. But then let's say that that issue goes ahead and gets updated. Once that issue gets updated, it may be that we don't want to be notified of that. Therefore, we can edit the notifications.

Click on **Delete** under the **Issue Updated** section. This will take us to the following screen:



The reporter will now be notified when an item is created but not when it's updated. The reporter is again notified when the item is assigned.

Summary

We've reached the end of our first chapter. In this chapter, we talked about what JIRA is, what it's used for today, why we should use it, and we talked about projects and how JIRA organizes its work into a project format. We learned how to create an account with Atlassian so that we can start using JIRA. We talked about creating and managing different projects in JIRA, and how to handle them. Lastly, we learned more about the specifics in the configuration of projects, such as screens, workflows, permissions, and notifications.

In the next chapter, we're going to talk about managing all of those work items in more detail so that we've got lots and lots to do, and JIRA will help us to achieve that.

2

Managing Work Items

In the previous chapter, we learned about what JIRA is, how to get started with it, and how to create a project in JIRA. In this chapter, we're going to learn about managing all of the work items that we have. Let's go ahead and get started.

In this chapter, we'll be learning about epics, stories, bugs, and tasks, what each of these things are, and how we would use and create them in an Agile project. Then, we're going to learn about issue type attributes. We're going to talk about adding and removing them, and making them really customized to fit the type of work that we're doing in JIRA. Then, we're going to learn about managing the items in our backlog, and then we're going to learn about our board (for those of us that do Scrum, we are going to feel very comfortable with the board), and how to configure it.

In this chapter, we will learn about the following topics:

- Introduction to epics, stories, bugs, and tasks
- Issue type attributes and adding and removing them
- Managing items in the backlog
- Creating and configuring our board

Introduction to epics, stories, bugs, and tasks

Epics, as you might imagine, are large stories. It's important that they have a distinct start and end, just like any story. We want them to be able to be completed, which means that they are going to need some sort of end. We like to think of epics as something that would span multiple sprints, versus a story, which would really be completed within a Sprint. Epics are not groupings of work items, and we'll take a look at that because that's a common thing that people do in a JIRA UX. We'll take a look at how we use components and labels to do this instead of epics. Epics will contain stories, bugs, and tasks, and they're really a grouping of those things, which is again a large story.

Stories are smaller than epics. Stories, bugs, and tasks are all the same level hierarchically; they're all things that could be prioritized against one another within our backlog. Stories are also known as **user stories**, and they're called that because the intent is that they will focus on users, we need to think about new functionality for whatever this is, and we need to make sure that we're continuing to think about the people that we're building it for. That's why they call them stories or user stories.

Bugs are what we call defects, and they occur when there's a problem. In JIRA, something will always prioritize against a new feature, and we know that we should take into account either, *do we want to take the time in this Sprint to fix something, or do we want to create something new?* We should be prioritizing those two things and pitting them against one another.

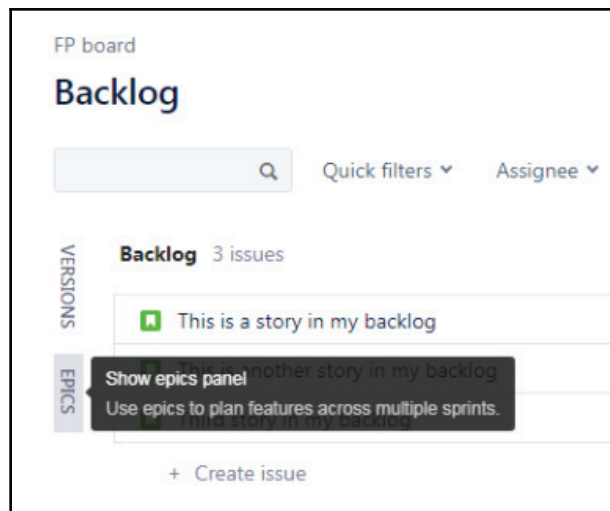
A subtask is something we'll take a look at quickly. If we have multiple people working on a story or a bug and we want to assign it a more granular level of detail, we can use a subtask.

Creating epics, stories, bugs, and tasks

To learn about epics, let's go to our projects. We created our **First Project** and **Second Project** in the previous chapter.

The following are the necessary steps to create an epic in a project:

1. Click on the **First Project** that we created in our JIRA account.
2. Within our project, we can see that we have the **Backlog** view. In this view, we have three stories. We can see on the side of the following panel that we have versions and we have epics. Click on **EPICS**:



3. We can see in the previous figure that we don't have any epics. We'll create an epic and we'll call this epic `My Test Epic`. In the summary, we can insert information regarding what the epic is about. Let's go ahead and create it. We need a summary, which we'll call `This is a summary`. We now have a test epic:

Create Epic

[Import issues](#) [Configure fields](#)

Project*
First Project (FP) ▾

Issue Type*
Epic ▾ ⓘ

Some issue types are unavailable due to incompatible field configuration and/or workflow associations.

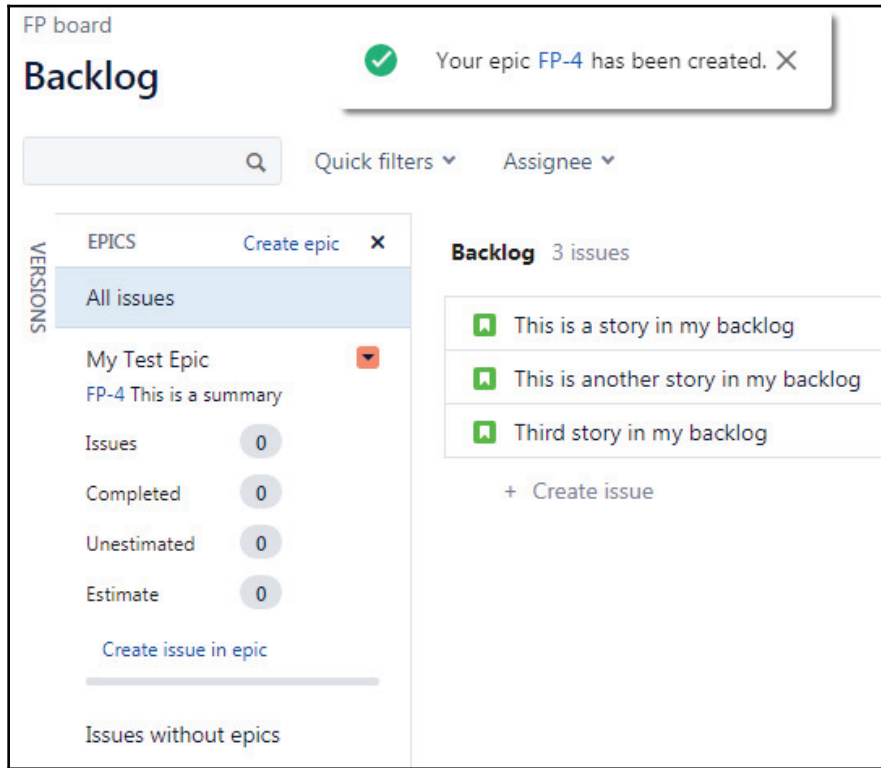
Epic Name*
My Test Epic

Provide a short name to identify this epic.

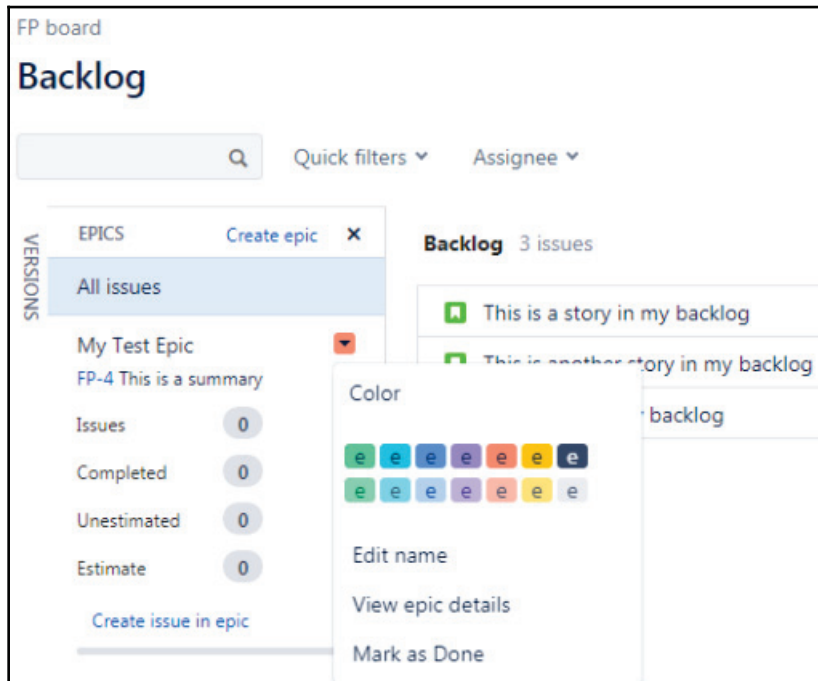
Summary*
This is a summary

Create another [Create](#) [Cancel](#)

Let's take a look at this epic. We can see that if we look at it on the left here that we can expand and contract the epic. It has a JIRA ID:



We can select the color that we want to use for it and we can edit the name, the epic details, and mark it as done:

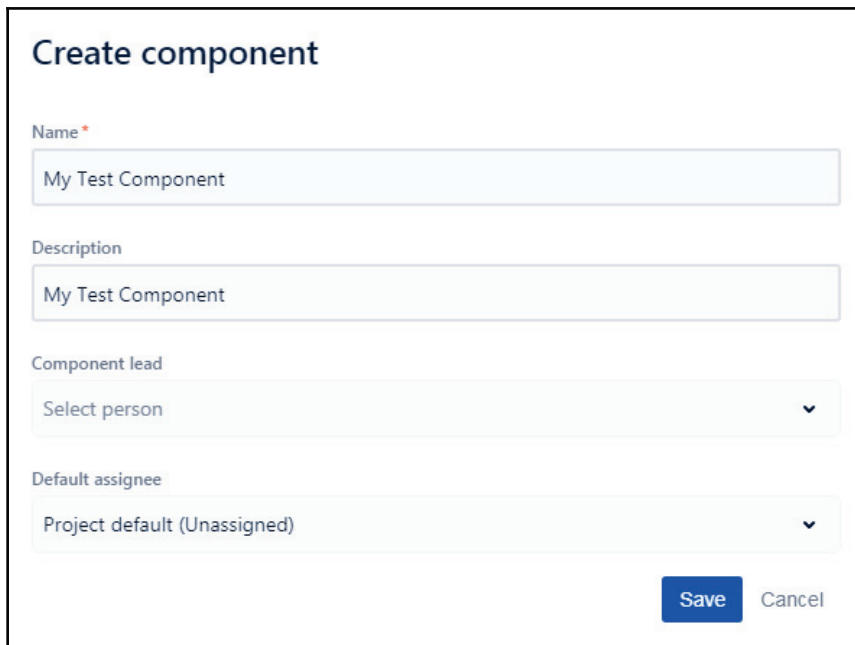


More importantly, underneath the epic, we can see how many issues or stories, bugs, or tasks are contained underneath this epic. We can also see how many are completed, unestimated, and how many are estimated. At the bottom, we can see a status bar.

We will create our backlog items and then *drag* those into our epic, and that's how we assign them to the epic. This epic is a story, it's a big story, but it's something that can be started and finished. It's important that this is not just a grouping, as that's not the intent of an epic.

If we want to group items in JIRA, that's a slightly different thing, and for that what we can do is we can use components to do that:

1. Select **Components** from the options on the left-hand side.
2. We'll create one component because we don't have one. Let's call this `My Test Component`, and, so we don't get an error again, we will use the same value that's in the description. We can select a **Component lead**, and also a **Default assignee**, who will basically be the same person that is assigned to the project by default:



Create component

Name *
My Test Component

Description
My Test Component

Component lead
Select person ▼

Default assignee
Project default (Unassigned) ▼

Save Cancel

We can use the component that we've created in order to group items together.

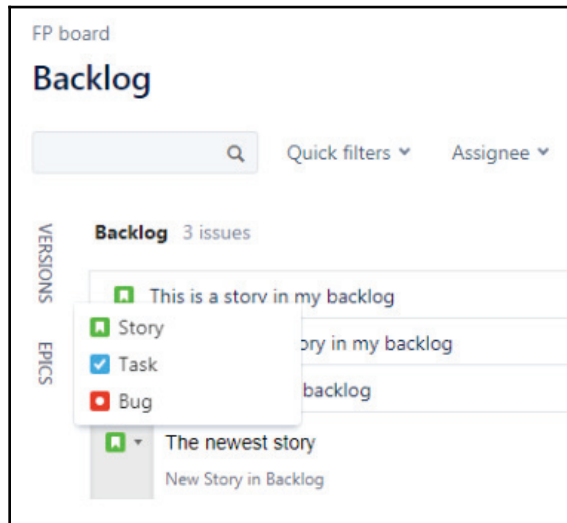
Let's have a look at how to create a story.

We can see that we have three stories. We'll go ahead and create a new one. We'll call this `The newest story`.

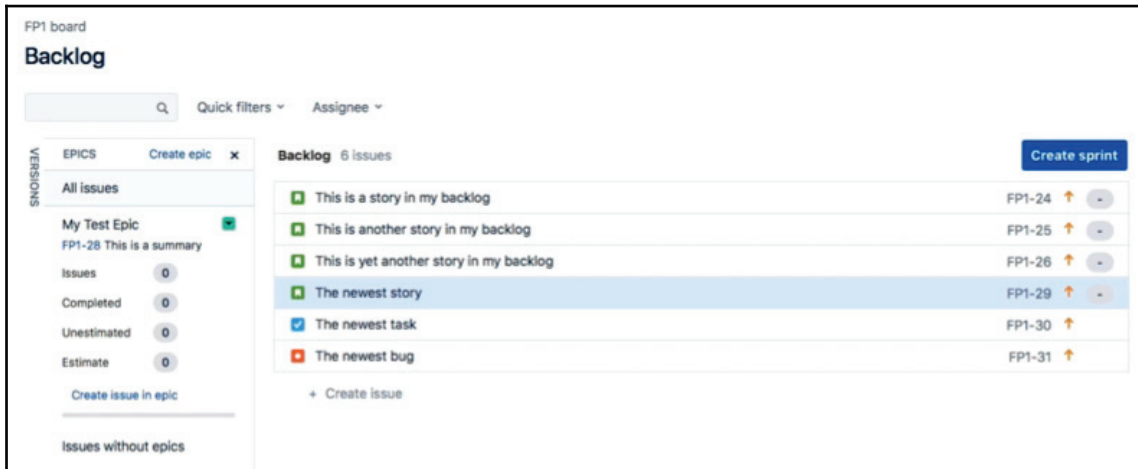
We can see that when we're actually creating the stories that we have the option to create a story, a task, or a bug, and that they're all ranked the same hierarchically within JIRA:



A story generally represents a piece of new functionality, a task is just something that needs to be done, and a bug is something that's broken that needs to be fixed.



We'll create a new story, create the newest task, and then we'll create the newest bug, and that way we have all three. We can take a look at them in the following screenshot:

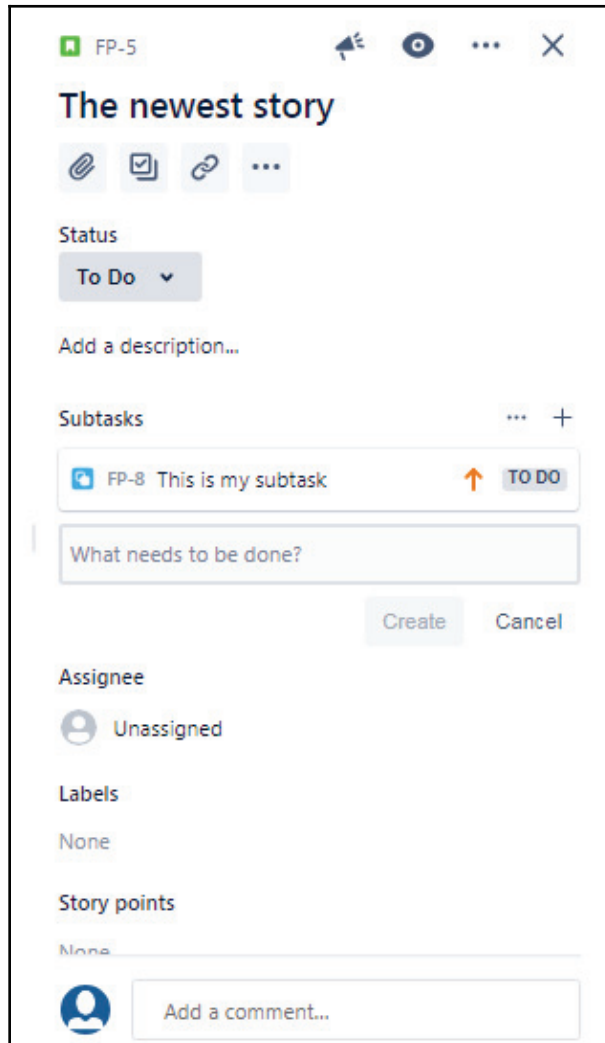


Backlog view

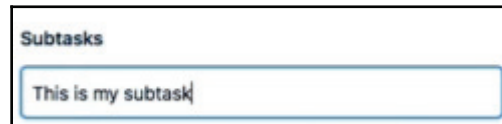
When we select one of these items, a preview pane on the right-side will show us details about it.

The other thing that we will look into is that when we select an item in this **Backlog** view, we can actually hit the *E* key on our keyboard, and it will prompt an edit. This is just a nice little shortcut. Another thing to note is that if we select an item and it's unassigned, we can actually hit the *A* button on our keyboard and it will prompt us to assign an item.

The last thing we'll look at in this particular section is how we create a subtask. If we've got our newest story on the preview pane, we can see that we've got the ability to add an attachment, an ability to link an item, to link to multiple items together and create dependencies, and then we have this creative subtask option:



Again, we might use the subtask if this story is going to need to be worked on by multiple people and if we really want to have more granularity around what each person's going to do. We can click **Create subtask**, and as we can see in the following screenshot, in the bottom right corner there's a prompt. We can say `This is my subtask`, and we can create it:



Create Subtask

Subtasks allow us to use multiple different items of work that exist underneath a story.

In this section, we learned about epics, stories, bugs, and tasks. In the next section, we're going to talk about issue type attributes. We're going to look at each of the different attributes that are available underneath the different issue types in more detail, and then we're going to talk about adding and removing them so that they fit our work better.

Issue type attributes and adding and removing them

In this section, we're going to cover issue type attributes, how to add them, and how to remove them.

First, we're going to learn what the different kinds of attributes are that can be assigned to stories, tasks, and bugs. Then, we're going to talk more about how we can customize them and adjust the screen so that we can only see the ones that we want to see.

The different type of work item attributes (stories, bugs, and tasks) are all JIRA work items.





The following are the attributes that are available to us:

- **Assignee:** This is the person that is assigned to use that item.
- **Attachments:** These are anything that need to be attached to that item to provide more clarity or show that it's been done.
- **Comments:** To add our views.
- **Component/s:** These are groupings, and are used to group items together.
- **Description:** This is an epic link.
- **Epic link:** We talked about how epics are effectively a large story in the last section. We would use this if we were looking at the story that the epic is associated with.
- **Fixed version/s:** This is a large group of functionality that is released at once. If the story, bug, or task is part of a version, it would be there.
- **Issue type:** This is a story, bug, or task.
- **Labels:** These are like components but are a way to group items across projects.
- **Linked issues:** This is a way in which we can create a dependency between different issues if something depends on something else, or if it's blocked by something else.
- **Priority:** This is slightly obvious. We can choose from high, medium, or low priority, and we'll use different priority statuses in JIRA.
- **Sprint:** Is the work item contained within a Sprint, and if so, which Sprint?

Let's take a look at JIRA and some of these work item attributes:

1. We've got the newest story that we created earlier. We'll press the *E* key on our keyboard, or we can use the three ellipses in the top-right-hand corner to go into edit mode:


The newest story


   

Status
To Do ▾

Add a description...

Subtasks ⋮ +

 FP-8 This is my subtask ↑ **TO DO**


Assignee
 Unassigned

Labels
cross-project-label

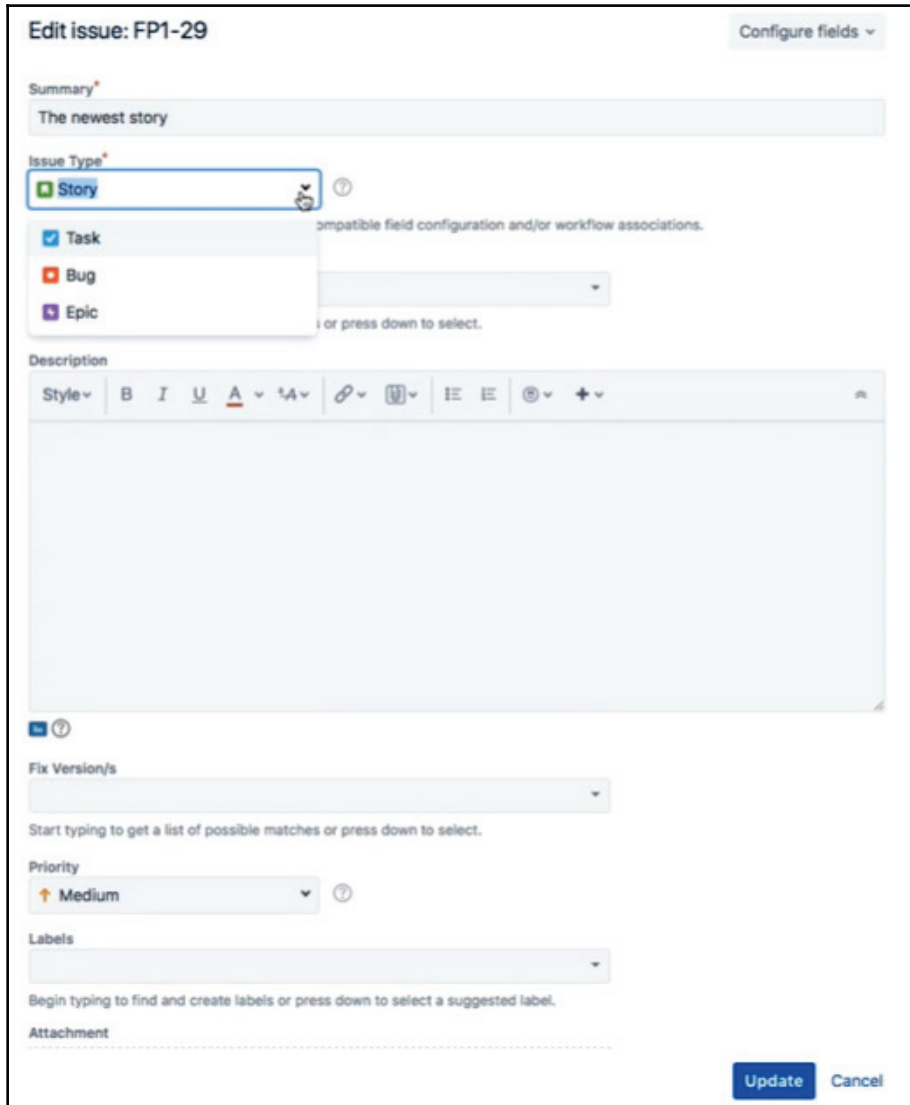
Story points
None

Components
My Test Component

Priority



- We're looking at issue **FP1-29**. This is our newest story. We can see in the following screenshot that as we move down the screen, we've got the ability to change the issue type from **Story** to **Task**, **Bug**, or **Epic**:



Issue type attributes and adding and removing them

In the last section, we created a component that we might remember. We can actually look for that component here. If we want to use this, we've got `My test component`, which allows us to group these items together:

Edit issue: FP-5
Configure fields ▾

Summary*

Issue Type*

Story ▾
?

Some issue types are unavailable due to incompatible field configuration and/or workflow associations.

Component/s

My Test Component ×
▾

Start typing to get a list of possible matches or press down to select.

Description

Style ▾
B
I
U
A ▾
²A ▾
🔗 ▾
🔒 ▾
☰
☰
😊 ▾
+ ▾
⋮

📖 ?

Fix Version/s

None

Update
Cancel

Grouping of items

The description has a rich text editor, which is nice. We've got **Fix Version/s**, which basically allows us to determine which release this is going to be in. A version is something that we eventually release. We also have a **Priority** here, and we have already talked about the priority being **High, Medium, Low**, and more. **Labels**, which are groupings across projects, are something we can actually look for just by selecting them, and we can also create new ones. If we use `cross-project-label`, we can see that we have one, we can go ahead and assign that label. We can use **Attachments** by dragging and dropping them, or we can browse. In **Linked Issues**, we can link the issue to another issue. There's a variety of ways we can link to things, either by using **blocks, is blocked by, clones, is cloned by, duplicates**, and more. Then, we can type in the issue here, and we can actually perform a search in real time. We can assign this issue to someone else or to ourself. We can assign the epic, and we can assign what sprint it should appear in, and we can also actually assign epics and sprints by dragging and dropping them:

Edit issue: FP-5 Configure fields ▾

Fix Version/s
None

Priority
↑ Medium ▾ ?

Labels
cross-project-label × ▾
Begin typing to find and create labels or press down to select a suggested label.

Attachment
 Drop files to attach, or [browse](#).

Linked Issues
blocks ▾

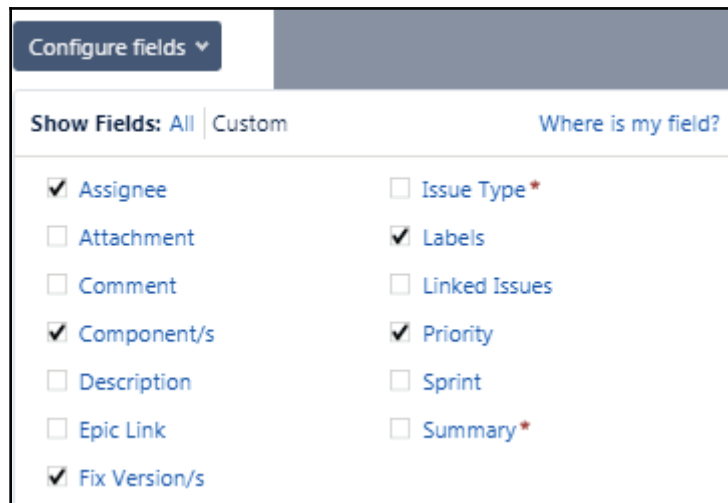
Issue
 +
Begin typing to search for issues to link. If you leave it blank, no link will be made.

Assignee
 Unassigned ▾
[Assign to me](#)

Epic Link
 ▾
Choose an epic to assign this issue to.

Then, we've got our rich text editor (again for comments) so that we can see the different fields that we have available to us. The other thing that we should note here is the **Configure fields** option in the upper right corner. Sometimes, we're not going to use a lot of these fields and we might use them all, but there's a chance we're not going to be using some of them.

3. If we select the **Configure fields** option, we can see that we have the ability to turn them all on and off, and we can even select the **Custom** field and uncheck the **Comment** field. When we save these changes, we'll see that this is the case for the items that we make go forward when we view them:



The screenshot shows a 'Configure fields' dialog box. At the top left is a button labeled 'Configure fields' with a downward arrow. Below it, there are two tabs: 'Show Fields: All' and 'Custom', with 'Custom' being the active tab. To the right of the tabs is a link that says 'Where is my field?'. The main area of the dialog contains a list of fields, each with a checkbox and a label. The fields and their checkbox states are: Assignee (checked), Attachment (unchecked), Comment (unchecked), Component/s (checked), Description (unchecked), Epic Link (unchecked), Fix Version/s (checked), Issue Type* (unchecked), Labels (checked), Linked Issues (unchecked), Priority (checked), Sprint (unchecked), and Summary* (unchecked). The asterisk on 'Issue Type' and 'Summary' indicates they are required fields.

Configure fields

4. Let's take a look at this preview pane in a little more detail; let's take a look at each of the items. We've got the ability to watch or unwatch an item:



Preview pane

This is helpful if we have multiple people on our project and we want someone to be notified if the status of this item changes. We can see that we're a watcher on this item, and this is really because it's assigned to us and we've reported it as well. We can see that it's actually unassigned, but it's been reported by us. That automatically makes us a watcher. If we remember our notifications workflow from the previous section that we went through, we've got our label that we created, we've got a status on this to-do in progress, and more, such as the ability to link an item, attachment, and subtasks. We can also perform a story point assignment, where we can look at the components, priorities, and reporters. We can also show more or less items. We can see that this actually contains time-tracking and things like subtask comments.

We get a lot of this through the preview pane, which is really convenient as we don't actually have to end up leaving the screen to do a lot of modifications. Some of these fields will not be editable once we pull them into a Sprint; once this story enters a Sprint, we're not going to be able to change some of these values. We'll take a look at this when we create a Sprint later.

In the next section, we're going to talk about managing those items in the backlog. We've got these items and we've assigned all of these attributes, so we're going to need to be able to manage them, prioritize them, and more.

Managing items in the backlog

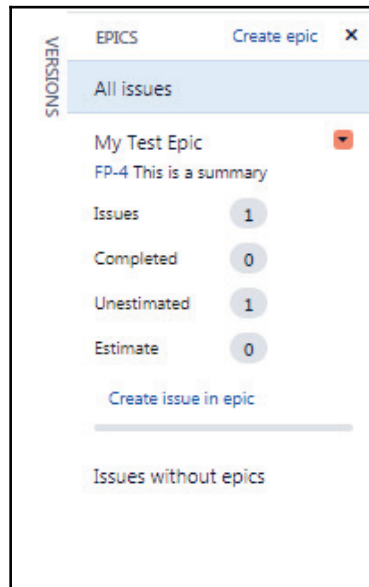
In this section, we're going to learn about managing the items in our backlog.

Here, we will learn about the following:

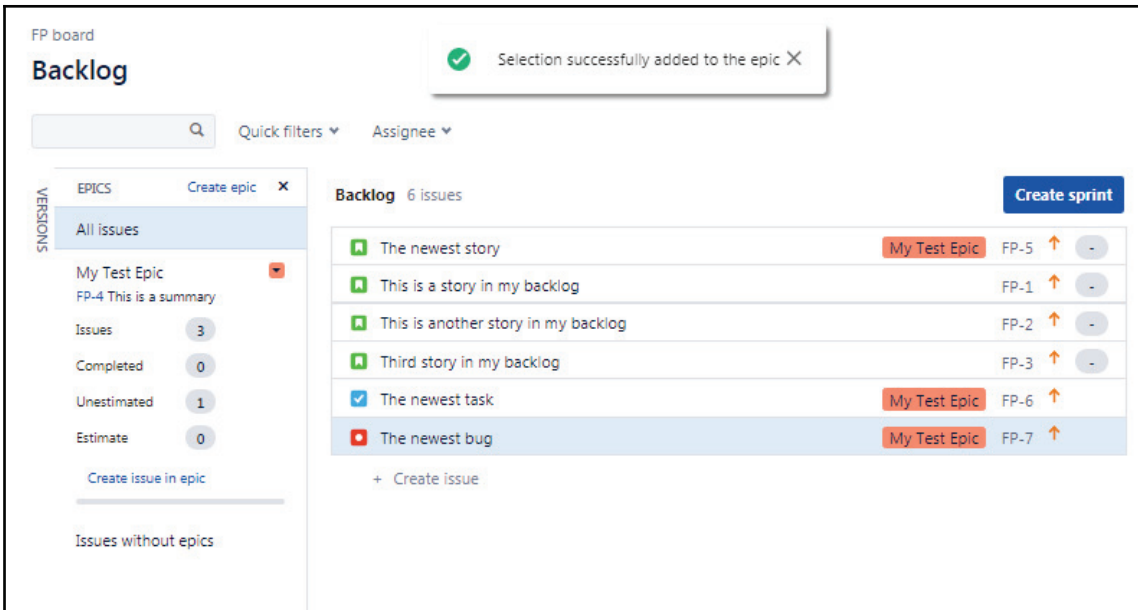
- How to create items
- How to prioritize items
- How to assign items into epics and versions
- How to create and use quick filters in our board view
- Separating work that's fully refined and ready for a Sprint

Let's go back to our backlog. We can see that we've got our items. It's really easy to prioritize these items in JIRA. All we have to do is basically drag and drop them. This means that we can move these items around in any way we want to, and that makes it really easy for our product owners to just move all this stuff around exactly how they want.

We made an epic previously, and as we can see in the following screenshot, we've got **My Test Epic**. We want to assign some items to this, maybe all of the newest items, which is fine. Let's take the first story (that is, **The newest story**), drag it over to our epic, and once we do that we'll get a message. We can also see that we have an issue assigning this on our epic:

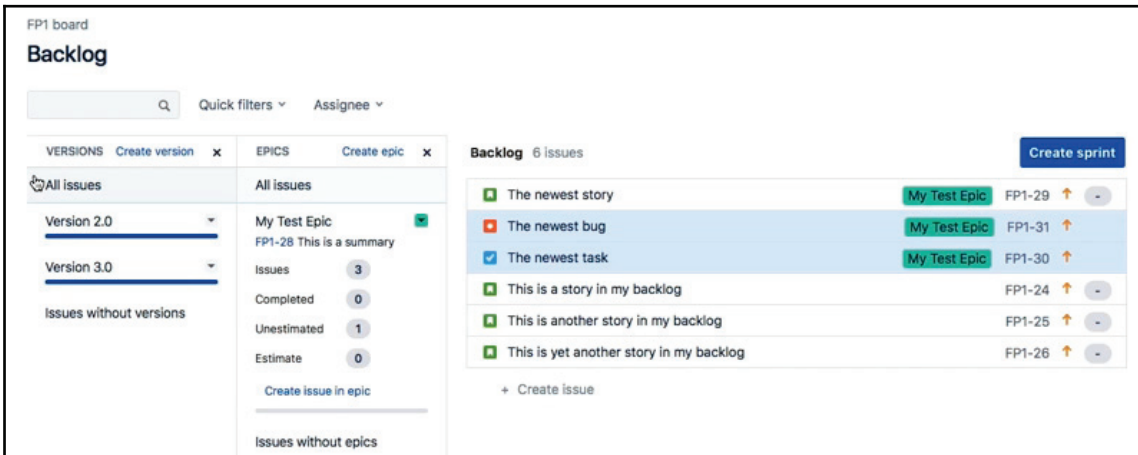


Let's go ahead and take the bug and the task. We can move these items over as well so that we have three items in our epic. We selected multiple items by just holding down the *Shift* key:



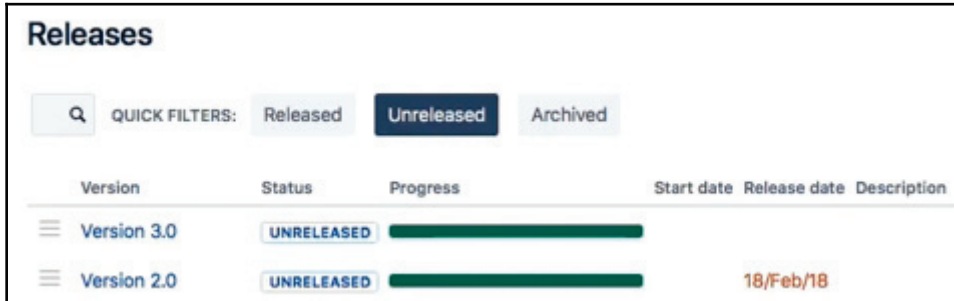
Adding items to Epic

Next, we've got what's called versions, and we can see here that this would be an increment of value that we would release to either our customers, to our market, or to something similar. In most cases, we'll use versions as a way to track the amount of work that's been completed in a month, in a week, or in a Sprint. So, here, we can go back and report on that:



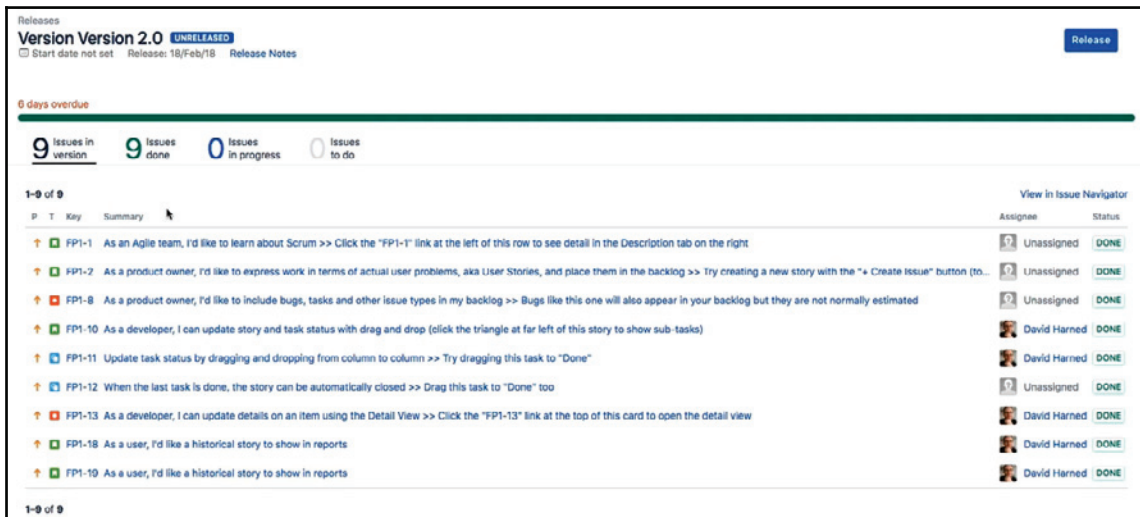
Managing items in the backlog

A version is a little bit different from an epic. We may have multiple epics inside a version, or we may just have one epic inside a version. We can see that in the preceding screenshot we have links to issues without versions, and issues without epics. These two may be related, but they're also not necessarily hierarchically connected. A version is a way for us to group work together and, as we can see, if we take a look at **Releases**, ultimately a version is something that we're going to release. As we can see in the following screenshot, we have version 2 and 3, which have been unreleased:



Releases

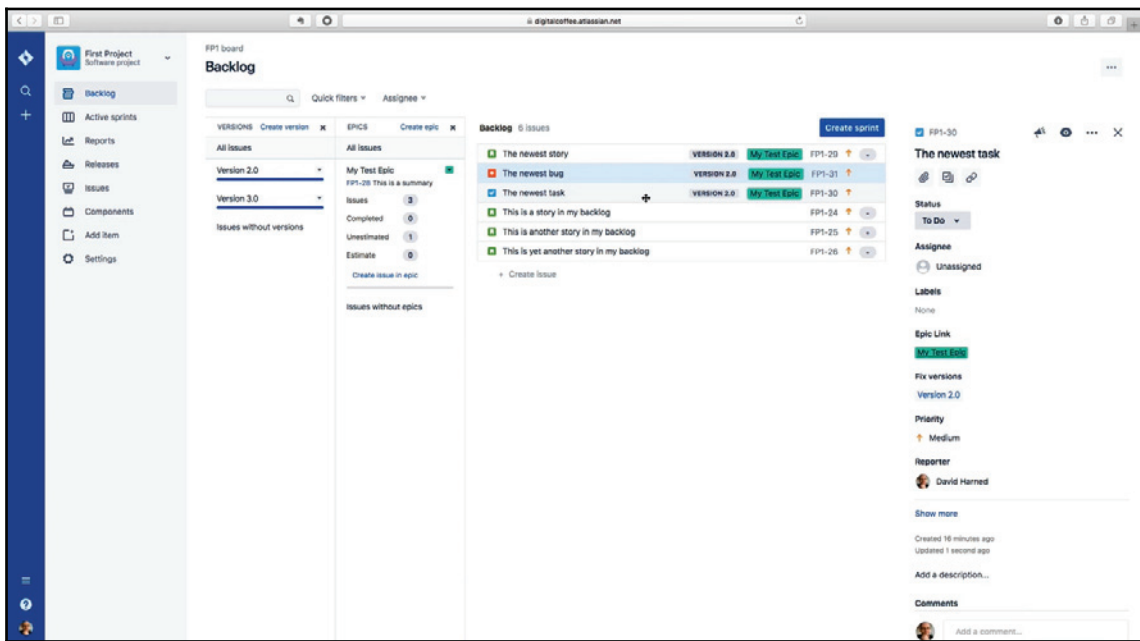
We can select these items and we can take a look at the items that appear underneath them. All default items that were in JIRA when we created this project are as follows:



Releases version

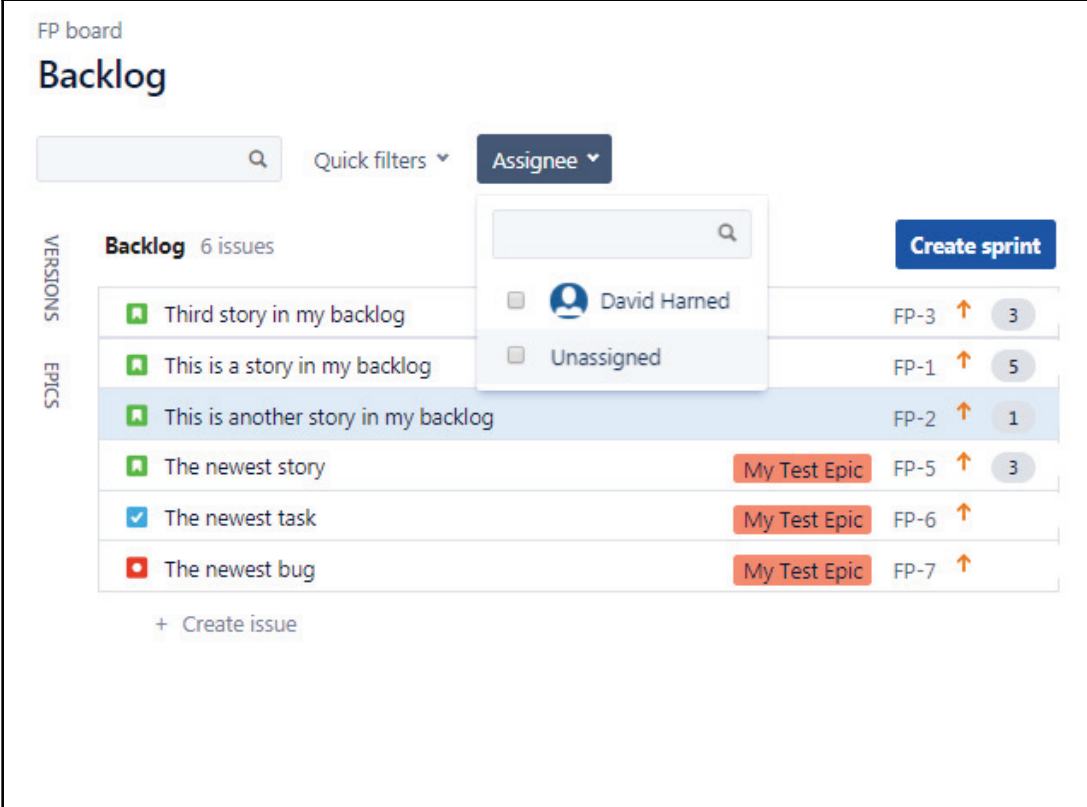
We can see that this gives we a nice view of things that are in the version, how much has been done, and how much is in progress. Once this is complete, we can hit the **Release** button, and we can actually release this version. That's what makes this version so powerful.

Going back to our backlog again, we can see that it's fairly easy to take the items and assign them to a version as well, and we can see that we've got some visual confirmation that these are in an epic. If we drag them over and put them inside a version, we're going to get that, too. We can see that this is contained inside version 2, and we can take the other two versions and do the same thing. The stories are in version 2, and they're also part of this epic:



Backlog view

Next, we want to take a look at **Quick filters**. These can be really powerful, especially if we're working together as a group with our team, and we've got to be looking at this backlog and moving quickly through. JIRA gives us some default quick filters. They only give us issues and items that have recently been updated. They also give us **Assignee**, which is also helpful, but let's say we want to take a look at anything that would be part of the component that we assigned earlier:

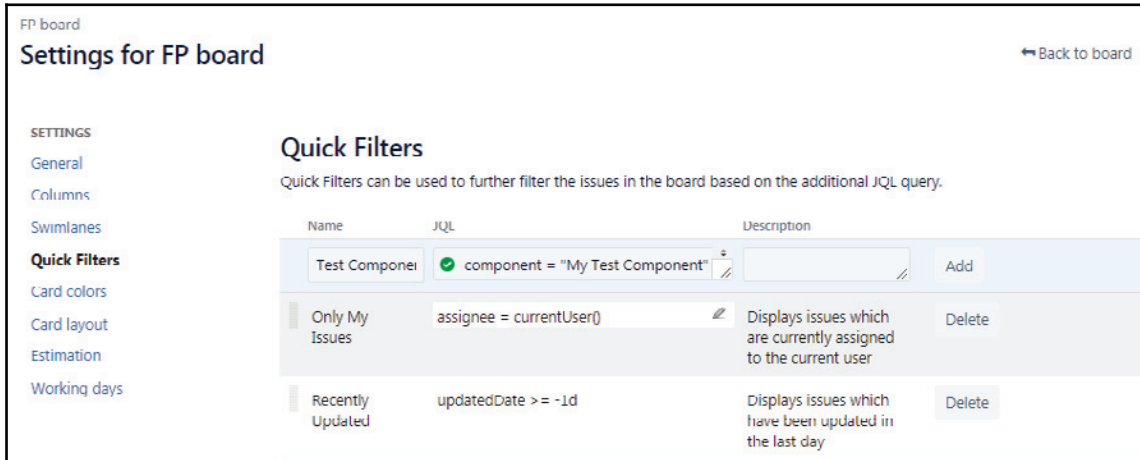


The screenshot shows the 'FP board' Backlog with 6 issues. A dropdown menu for 'Assignee' is open, showing 'David Hamed' and 'Unassigned'. The issues are grouped by 'My Test Epic'.

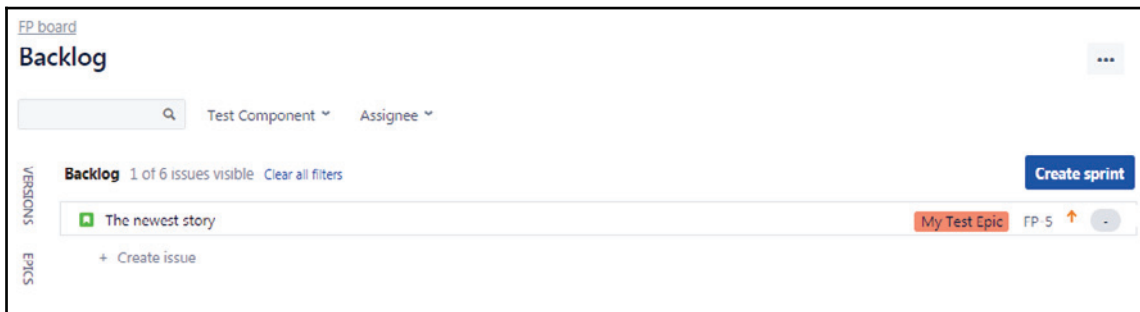
Issue	Assignee	Priority	Count
Third story in my backlog	Unassigned	FP-3	3
This is a story in my backlog	Unassigned	FP-1	5
This is another story in my backlog	Unassigned	FP-2	1
The newest story	My Test Epic	FP-5	3
The newest task	My Test Epic	FP-6	1
The newest bug	My Test Epic	FP-7	1

We had this component that was a grouping of items that we put together, which was called **My Test Component**. Why don't we go back to our **Backlog**? We're going to go to the top corner of the screen and select **Board settings**. We'll take a look at a lot of these board settings later in the next section.

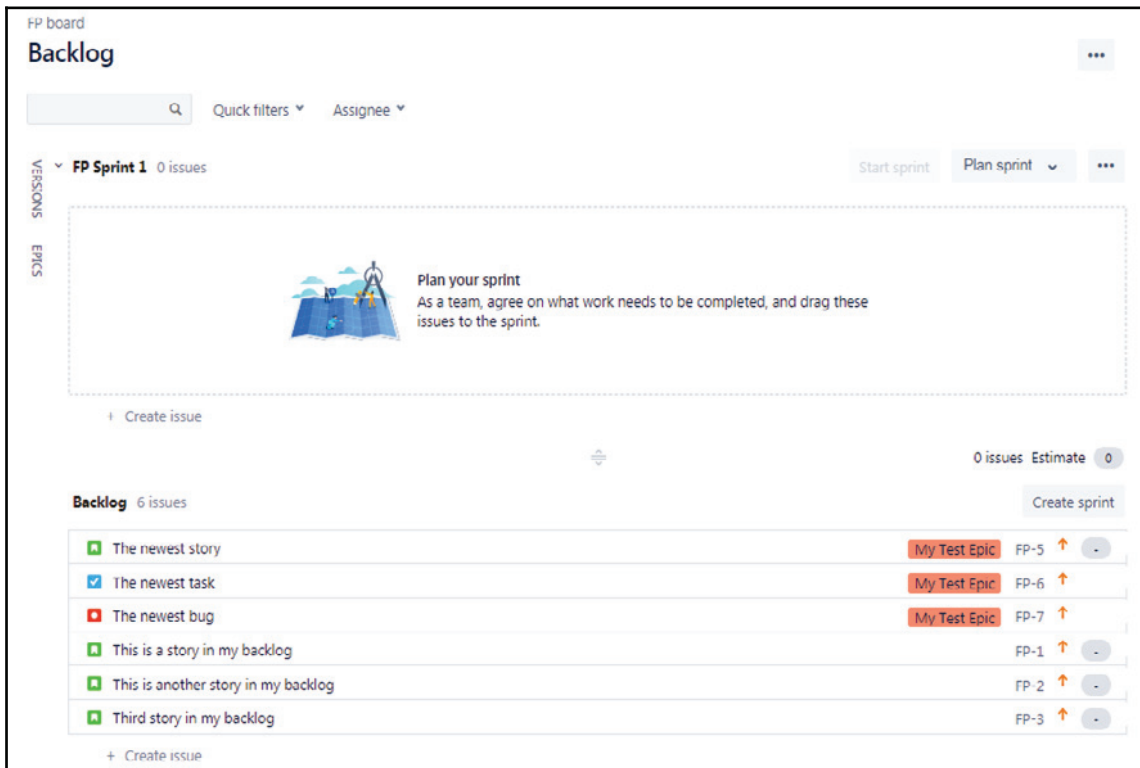
We're going to go under **Quick Filters** and call this `Test Component`. In our query for this, our **JQL** is going to be `component= "My Test Component"`. We can also put a description if we want to. Next, we'll hit the **Add** button, and we'll have created a quick filter called `Test Component`:



If we come back to the **Backlog** again, we can see that we have a **Test Component** query. If we select this, only the items that have components assigned to them should appear. We assigned that component as **My Test Component** earlier to the newest story when we were in the **Edit** window for that story:



The next thing we want to take a look at is the creation of a ready Sprint. Click on **Clear all filters** so that we can see all of our items. We can see the **Create sprint** button, where we can create a Sprint. We're not going to run it, but we can create a Sprint, and we can basically use that as a container to help us when we have work in our backlog that's ready to go. We call that refined. We've got work here that's refined, which means that it meets our definition of ready for the team that's coming into a Sprint. This means that will have acceptance criteria, it will be sized with some sort of a value, and it will have all the information required for us to to fix or build something:



We can take the three newest items, as shown in the following screenshot, and we can drag them straight up and place them into a Sprint. We can call this Sprint Ready:

FP board

Backlog

Quick filters ▾ Assignee ▾

Ready 3 issues Start sprint Plan sprint ▾ ⋮

VERSIONS

EPICS

<input type="checkbox"/>	The newest story	My Test Epic	FP-5	↑	-
<input checked="" type="checkbox"/>	The newest task	My Test Epic	FP-6	↑	-
<input type="checkbox"/>	The newest bug	My Test Epic	FP-7	↑	-

+ Create issue

3 issues Estimate 0

Backlog 3 issues Create sprint

<input type="checkbox"/>	This is a story in my backlog	FP-1	↑	-
<input type="checkbox"/>	This is another story in my backlog	FP-2	↑	-
<input type="checkbox"/>	Third story in my backlog	FP-3	↑	-

+ Create issue

Then, what we can do is we can use that fully refined work, not only in our next Sprint, but as the team increases in velocity, to actually pull stories from this section into our Sprint so that the team has work as they accelerate. We'll look more at that as we get into running a Sprint.

In the next section, we're going to talk about treating and configuring the board. We've got our backlog ready to go, we've got issues that have been created, and they've been assigned to epics and versions. We know what all the attributes are underneath those things.

Next, we're going to take a look at the board, which is what we're going to use when we're running a Sprint, which means that we're preparing to run our first Sprint.

Creating and configuring our board

In this section we're going to talk about creating and configuring our board.

In this section we're going to talk about the following:

- Differences between a virtual scrum board and an actual scrum board
- The columns that represent workflow states and how to modify a workflow
- Swimlanes
- Estimation options—time versus story points

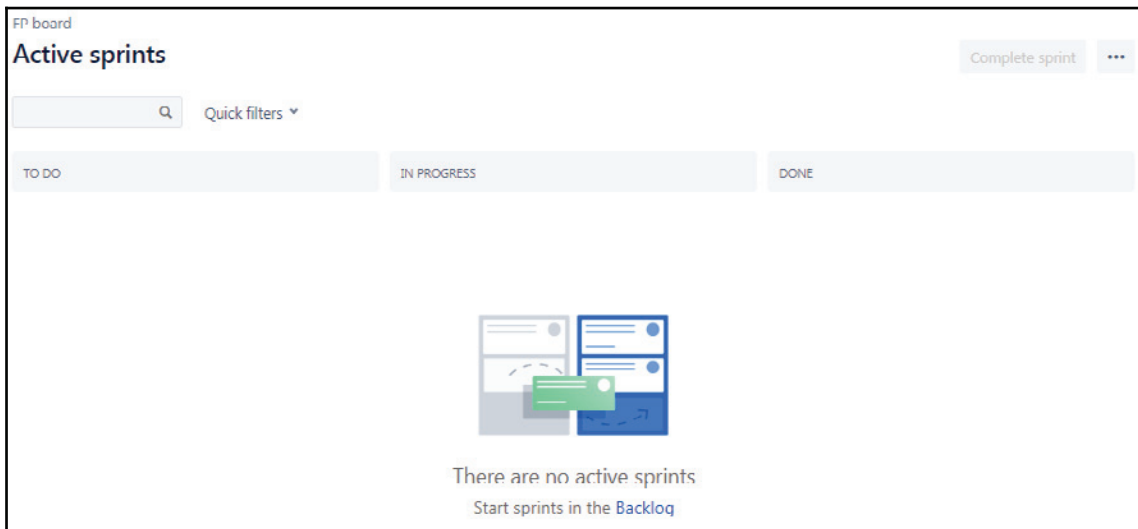
Let's go ahead and go over to JIRA. We can see what we've done here is we've actually taken our items that were not in our **Ready** Sprint. They have not been refined, and we have taken them and pulled them into a **Sprint 0**, which is a test Sprint. This will eventually be the Sprint that we'll put on our Sprint board, but in the meantime, we have to get our board ready. We can see that what we've done here is we've actually assigned story point values to each of these stories. We will be able to see them in the screenshot in gray, and we can also see them in the preview pane as well:

The screenshot displays the JIRA Backlog view for an 'FP board'. It is organized into swimlanes. The top swimlane is 'Sprint 0' (3 issues), which contains three stories: 'Third story in my backlog' (FP-3, 3 points), 'This is another story in my backlog' (FP-2, 1 point), and 'This is a story in my backlog' (FP-1, 5 points). Below it is the 'Ready' swimlane (3 issues), containing 'The newest story' (My Test Epic, FP-5, 3 points), 'The newest task' (My Test Epic, FP-6, 6 points), and 'The newest bug' (My Test epic, FP-7, 7 points). At the bottom is an empty 'Backlog' swimlane (0 issues). A preview pane on the right shows details for the story 'This is another story in my backlog', including its status 'To Do', assignee 'Unassigned', and reporter 'David Hamed'.

Backlog view

The following screenshot represents our **Active sprints**, and this is eventually going to be our board. If we hit **Start sprint**, these items will appear on this active Sprint board, but let's just take a look at this board before we start a Sprint.

A physical Scrum board would be a white board with columns on it. Each one of those columns would represent the different steps in our workflow:



And we can see, in its simplest form, we've got things that need to be done, we've got things that are in progress, and we've got things that have been completed.

Our **Quick filters** are also available for us. We can see that we have the ability to take a look at this, and a lot of times what we want to do is actually set up all of the different team members that are going to be working on stories in this Sprint and put them across the top. Then, as we're going through a daily Scrum, we can actually click on each name and look at just the items for that person and try and figure out if there are any impediments. This is the role of the Scrum master as we move through a Scrum Sprint.

Therefore, we need to add a testing step in the workflow so that things should be in **TO DO**, and then they'll be **DONE, IN PROGRESS**. Once they've been worked on, they need to be tested before they're considered done, and that way we can maintain a high level of quality. We have lots of testers in our Scrum team, and this is going to work out just fine because we can actually move a card into that testing column and we can have one of the testers help us with testing that. If we move into the upper right corner here, we will see three ellipses (...). Let's go into **Board settings** and take a look at this. We were in here previously, setting up a quick filter for our **Test Component**. Look under **General** settings for our board, as shown in the following screenshot:


General and filter

The Board filter determines which issues appear on the board. It can be based on one or more projects, or custom JQL depending on your needs.

General


Board name
FP board

Administrators
David Harned (admin)

Location
 First Project (FP)


Filter

Saved Filter
Filter for FP board
[Edit Filter Query](#)

Shares
 **Project:** First Project
[Edit Filter Shares](#)

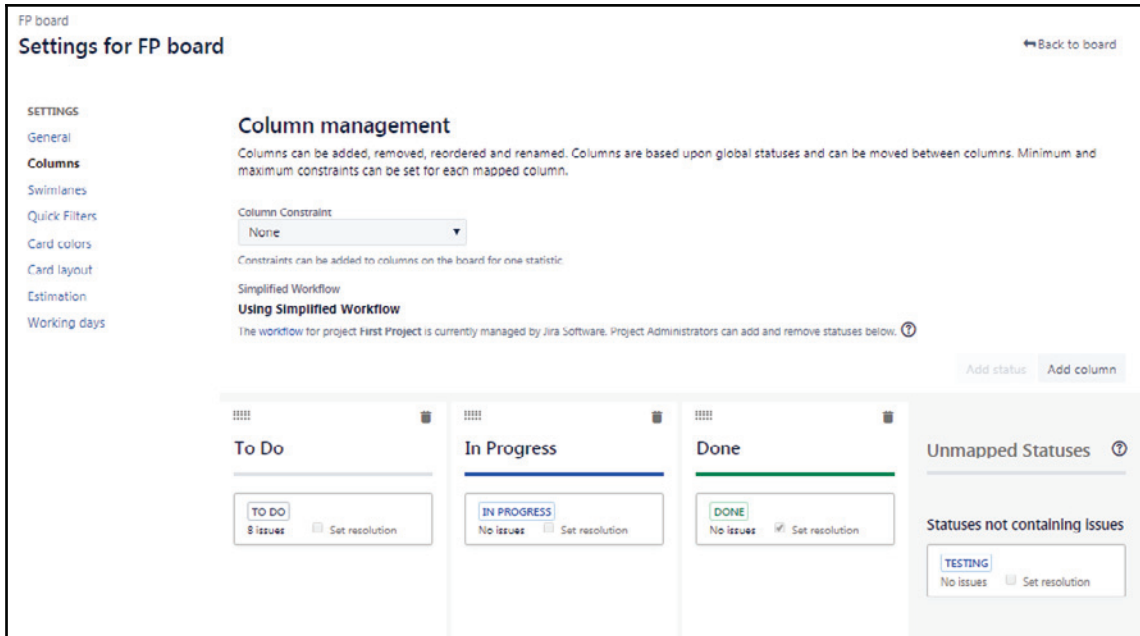
Filter Query
project = FP ORDER BY Rank ASC

Ranking
Using Rank

Projects in board
 First Project
[View permission](#)

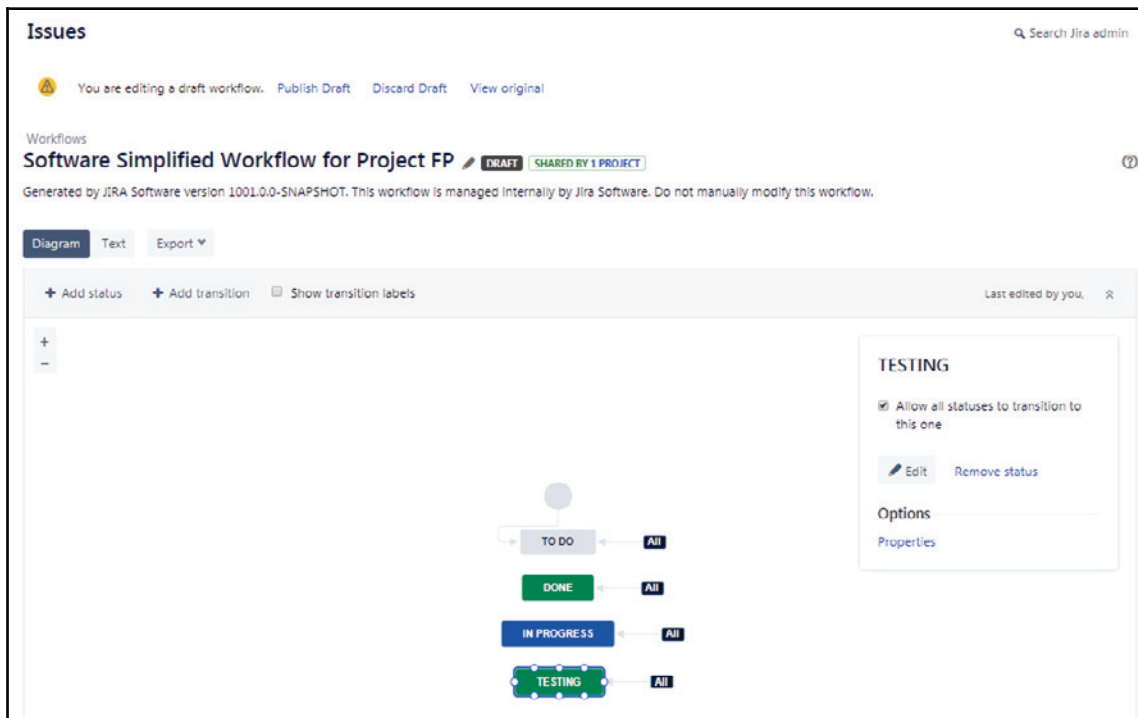
We can see that we have things like our **Board name**, which we can change regarding whether it appears here, the **Administrators**, and the location. We can actually create queries and generate boards from our queries, but we're just using the JIRA default for this project. We can see that in the preceding screenshot, we have the project, we have the query, and we can see that we've actually got multiple projects that we can put into a board as well, but we're going to leave these settings as they are.

Next, we'll take a look at **Columns**:



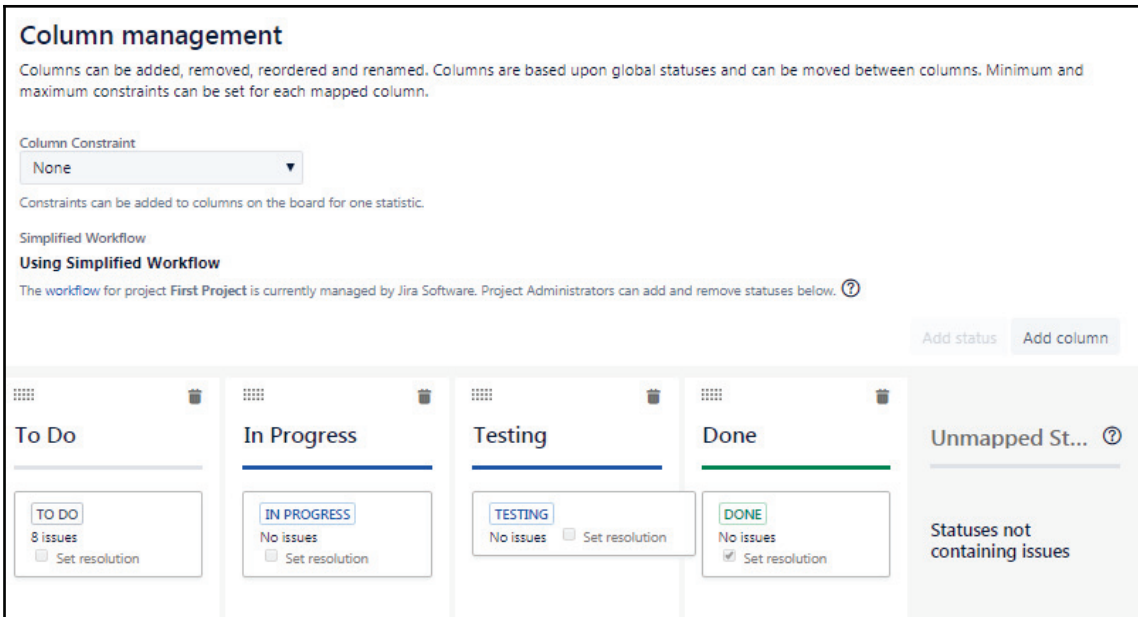
Settings for FP board

This is the thing that we wanted to modify. We want to add a testing step in my workflow. In order to add a new column, we'll need another status, all right? We've created this **TESTING** status in my workflow, as we can see in the preceding screenshot, and the way we add another status to this is by going to the **Workflow**. Remember the workflow from a previous section that we did? We can take a look at these workflows by selecting the edit icon:



Issues

We can see here that what we've done is we've actually created a **TESTING** step. Any step can transition into this step. Basically, we've got this **TESTING** step, and we are allowing all statuses to transition to this one; we have this in my workflow. Let's go ahead and go back to our project. We're going to go back into **Issues**, go to **Settings**, go to **Projects**, and we're going to go back to our **First Project**. We should be at the board for our **First Project**. We've created that step in the workflow. If we go back into our **Board settings**, in the columns, we can add a column and we can say that we want to call this column **Testing**. We can see that this column won't show on the board without a status, but that's okay because we created that status in the workflow. We can move that status into the column we just created:

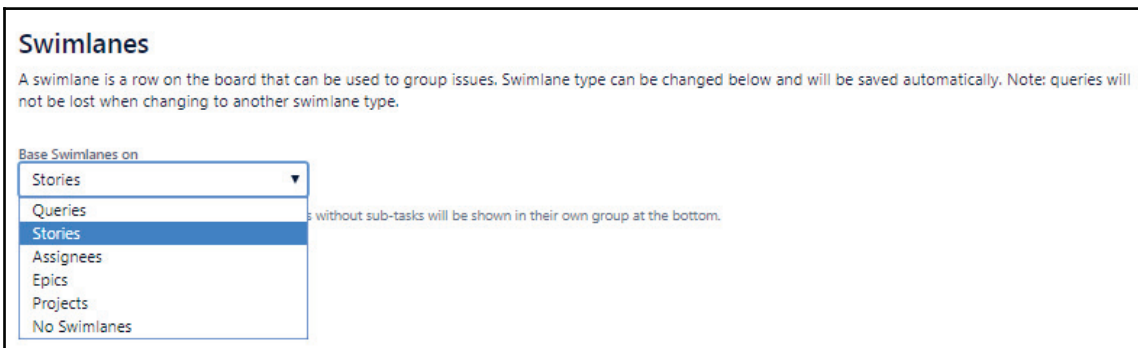


Column management

We have the **To Do**, **In Progress**, **Testing**, and **Done** columns. If we go back to the board, we'll see that we have a **TESTING** column.

Next, we can take a look at **Swimlanes**.

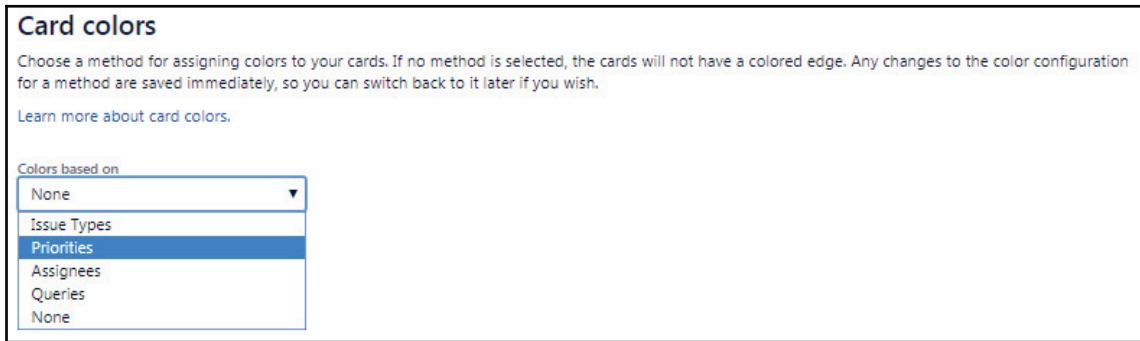
Swimlanes actually cut across a board, and we can use them in a variety of different ways. JIRA gives us some options to do that:



Swimlanes

In most cases, what we might do is for **Assignees**, **Epics**, or **Stories**, if we've got stories with subtasks underneath them then our subtask will be what is shown on the board, and we can expand and collapse the stories in **Swimlanes**. We'll leave it at **Stories** for now.

We've already talked about **Quick filters**. We've got some capabilities to modify **Card colors** based on the options shown in the following screenshot:



Card colors

In other words, if we've got a high-priority card, because the priority is set to high/critical, we can have it change color. We can also change the items that appear on our card, and we can show three additional fields. This allows us to pick what those three additional fields might be, but for now we'll leave them as their default settings:

Card layout

Cards can be configured to show up to three extra fields.

Backlog

Field Name

[CHART] Date of First Response
▼
Add

Active sprints

Field Name

[CHART] Date of First Response
▼
Add

Days in column

✕

Show a visual indicator on each card that represents the time spent in the column. This can help identify slow moving issues.

Card layout

Then, we've got **Estimation**. What we do with Scrum is we measure my velocity in story points. We don't really use time, as we found that relative estimation is much more effective than time estimation. However, lots of teams still use time to perform estimations:

Estimation

Issues can be estimated when in the Backlog to get an idea of how much work is being committed to in a sprint. [Read more about estimation and tracking.](#)

Estimation Statistic

Story Points
▼

Estimate issues in the Backlog by entering values for **Story Points**. Your velocity from sprint to sprint will be measured against these estimates.

Time Tracking

None

Issues will burn down their **Story Points** value upon completion.

Remaining Estimate and Time Spent

Track time against issues using Jira's **Remaining Estimate** and **Time Spent** fields.

Estimation

If we do use time and we're burning down hours, this would be where we would want to make that change. We would want a **Remaining Estimate and Time Spent**, we'd select the item, and then we can perform our estimation on the original time estimate.

Finally, **Working Days**, as we might imagine, is really about setting the working days for the group that's going to be using this board:

Time Zone

Region:
System default ▼

Time Zone:
(GMT+05:30) timezone.zone.asia.c ▼

Standard Working Days

Every

- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday
- Sunday

Non-Working Days

Dates:
No dates

Time Zone

We've configured all of this, we've got our board ready to go, and we've got our Sprints with items in them. We think we're probably about ready to start a Sprint, and that's what's going to happen in the next chapter.

Summary

Let's talk about what we learned. We talked about what epics, stories, bugs, and tasks are, and how to use each of them, and why we use one versus the other. We learned about the different work item attributes that are available to us, free issue types, and then learned what they are and how to add or remove them so that we can configure JIRA to meet the needs of our team. Then, we learned how to manage the items that are in our backlog, prioritization, assigning the edit, assigning to epics and versions, and more. Finally, talked about what a board is, how to configure that board so that the workflow matches our workflow, and how to effectively manage those items as they go through a Sprint.

In the next chapter, we're going to go ahead and start our project.

3 Running Your Project in JIRA

In *Chapter 1, Get Started with Creating Your Project*, we figured out what JIRA is and how to get started with creating projects. In *Chapter 2, Managing Work Items*, we talked about managing work items and what the different work items are, which we went into a lot of detail over.

In this chapter we're going to talk about our project, and how we need to use JIRA to manage the project while it's running.

In this chapter, we will learn about the following topics:

- Creating and starting a Sprint
- The daily Scrum
- Smaller stories or tasks
- Closing the Sprint—the Sprint Report

Creating and starting a Sprint

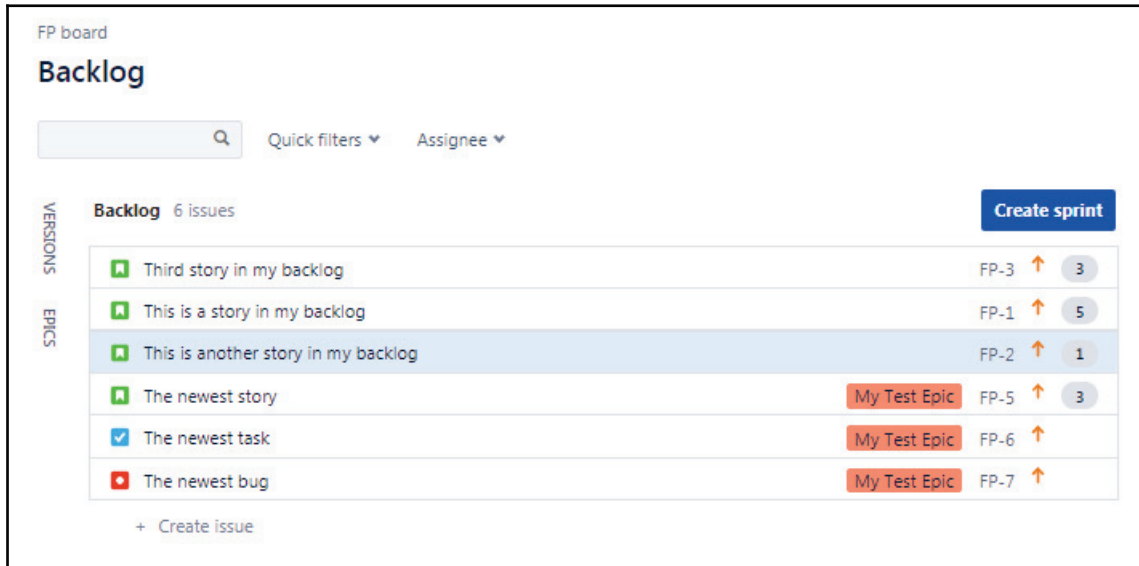
In the last chapter, we talked a lot about epics, stories, bugs, and tasks in a lot of detail, so if you want to talk more about those things or look deeper into them, then you should go back to *Chapter 2, Managing Work Items*, and look at them again. In this section, we're going to talk about creating and starting a Sprint.

In this section, we will cover the following:

- The backlog view, which allows us to prioritize the stories, tasks, bugs, and things that are in there
- Filtered views of the backlog
- Creating Sprints to have containers for the work

- Story point estimation
- Sizing the team commitment
- Adjustment to a Sprint

Let's go ahead and flip over to JIRA. We're going to look at this project. This is the **Backlog** view, and we can see that we have some stories in the backlog, as shown in the following screenshot:



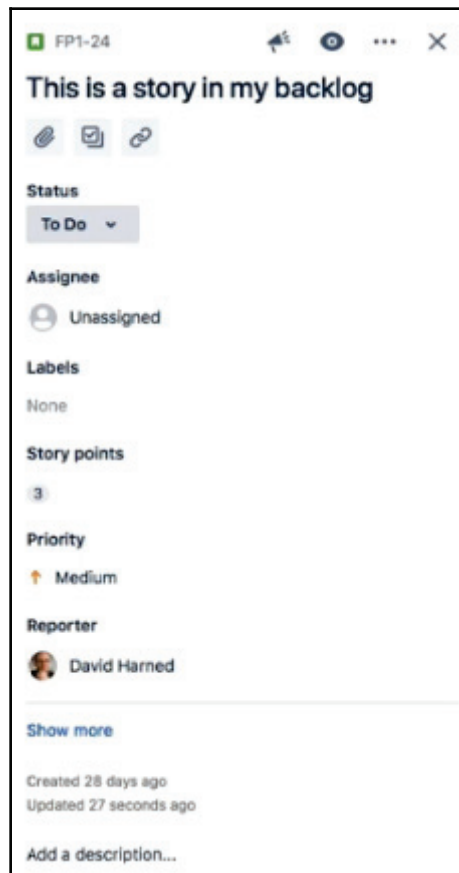
The screenshot shows the JIRA Backlog view for a project named 'FP board'. The title 'Backlog' is prominently displayed. Below the title, there is a search bar and two dropdown menus labeled 'Quick filters' and 'Assignee'. On the left side, there are two vertical tabs: 'VERSIONS' and 'EPICS'. The main content area shows a list of six issues. The first three are stories, and the last three are a task and a bug. Each issue has a priority level and a story point value. A 'Create sprint' button is visible in the top right corner. At the bottom left, there is a '+ Create issue' link.

Issue Type	Issue Title	Priority	Story Points
Story	Third story in my backlog	FP-3	3
Story	This is a story in my backlog	FP-1	5
Story	This is another story in my backlog	FP-2	1
Story	The newest story	My Test Epic FP-5	3
Task	The newest task	My Test Epic FP-6	
Bug	The newest bug	My Test Epic FP-7	

Backlog view

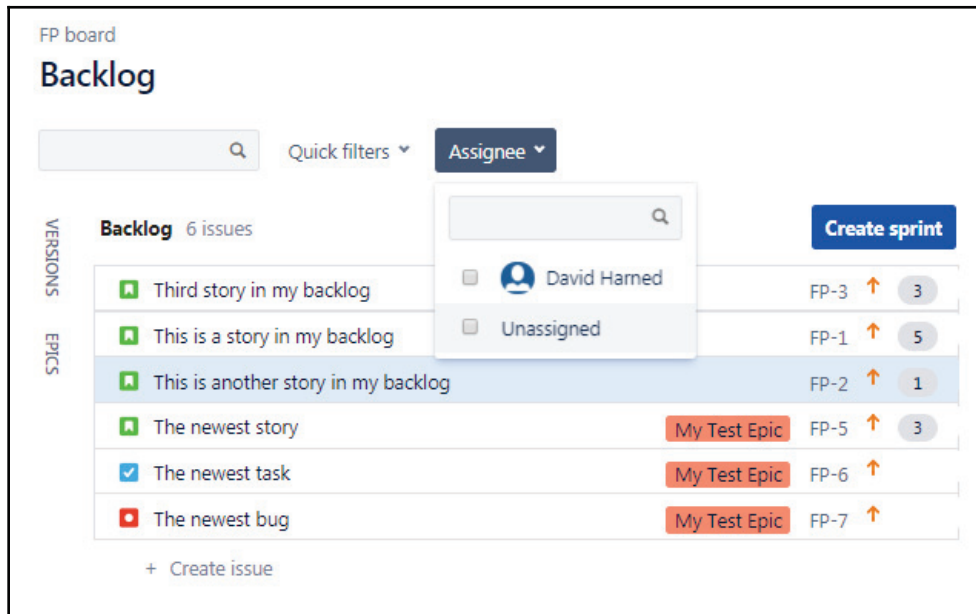
As we can see from the preceding screenshot, we have four stories, a task, and a bug. We can prioritize these items by just dragging and dropping them, and we can move them around and put them in whatever priority order we like. The most important item will be on the top.

We can see in the preceding screenshot that these stories have point values. We can see on the right-hand side that we have our first story, which has a story point value of **3**, and that the one below it has a story point value of **1**, and so on. Remember these story points and then refer to the doubt, effort, and complexity contained within this item. We have the ability to edit these values by selecting an item, and then going into the preview pane on the right:



Creating and starting a project

We can edit some of these values, whether it's the **Status**, **Assignee**, **Labels**, **Story points**, and more, and then we've got the ability at the top to filter by **Assignee**, as well as some other **Quick filters**, which has **Only My Issues**, **Recently Updated**, and more in its dropdown. We can see in the following screenshot that there's only one **Assignee**. In a real life scenario, we would have our whole team listed:



We have this backlog with some items in it, and we want to do a couple of things. First of all, we've got what's called a definition of ready, which includes any of our items that have been fully refined, which means that all of the questions we had have been answered, all of the understanding of what this thing is have been defined, and they meet what's called a definition of ready for our team. Our definition of ready could have acceptance criteria in it. We understand when the product owner thinks the item is done, and other criteria will be met in order for it to be ready. We want to have, ideally, at least two Sprints of ready work that's ready to go at any time.

Let's do that first. We'll create a Sprint, which we're going to call Sprint Ready. By clicking on the ellipses, we can edit this Sprint and name it *Ready*, and include the **Sprint goal** as Work that is fully refined, as follows:

Edit sprint: FP Sprint 1

Sprint name: *

Sprint goal:

Work that is fully refined

After we update this, we have a `Ready` Sprint. We can see the first three stories in our backlog, and they look like they're ready to go. Drag them up into our `Ready` Sprint, which has items in it that are fully refined and ready to go:

FP board

Backlog

Quick filters ▾
Assignee ▾

▼ **Ready** 3 issues

Work that is fully refined

⋮

VERSIONS
EPICS

<div style="display: flex; align-items: center;"> ■ Third story in my backlog </div>	FP-3 ↑ 3
<div style="display: flex; align-items: center;"> ■ This is another story in my backlog </div>	FP-2 ↑ 1
<div style="display: flex; align-items: center;"> ■ This is a story in my backlog </div>	FP-1 ↑ 5
+ Create issue	

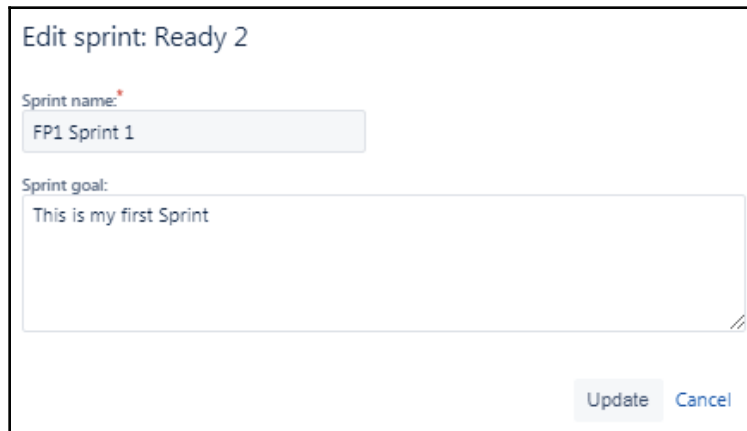
3 issues Estimate 9

Backlog 3 issues

<div style="display: flex; align-items: center;"> ■ The newest story My Test Epic </div>	FP-5 ↑ 3
<div style="display: flex; align-items: center;"> ■ The newest task </div>	FP-6 ↑
<div style="display: flex; align-items: center;"> ■ The newest bug </div>	FP-7 ↑
+ Create issue	

What we want to do is create a Sprint that we're actually going to execute.

Let's create another Sprint. We'll call this one `FP1 Sprint 1` and include `This is my first Sprint` as the **Sprint goal**:



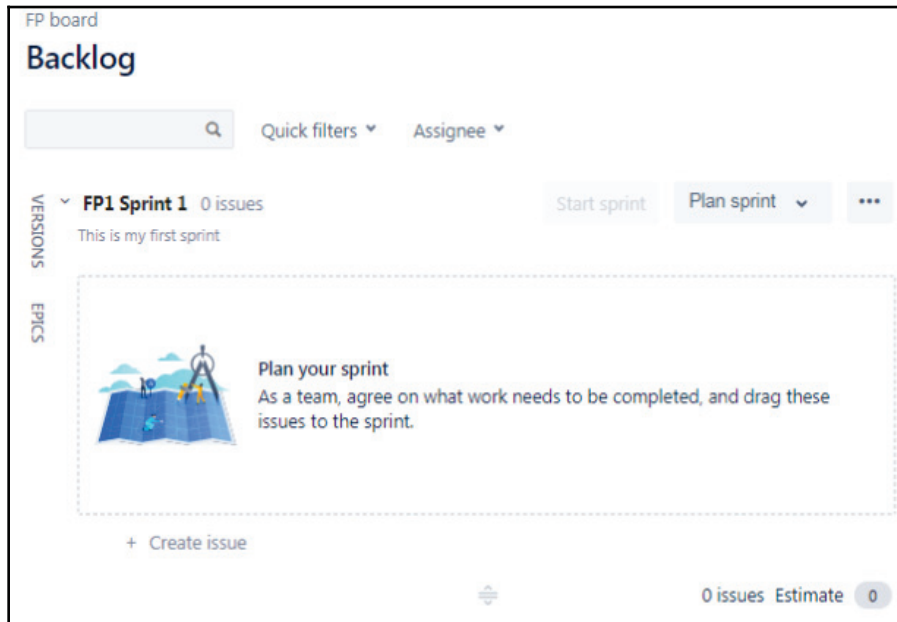
Edit sprint: Ready 2

Sprint name:

Sprint goal:

Update Cancel

We can actually take this item and move it up and put it at the top of the list as shown in the following screenshot. Then there is **Backlog** down that represents anything else that is in this list of things that we might want to eventually use:



FP board

Backlog


Quick filters ▾ Assignee ▾

VERSIONS

▼ **FP1 Sprint 1** 0 issues Start sprint Plan sprint ▾ ⋮

This is my first sprint

EPICS

 **Plan your sprint**
As a team, agree on what work needs to be completed, and drag these issues to the sprint.

+ Create issue

0 issues Estimate 0

We're going to take two of them and pull them up indoor Sprint as follows:

The screenshot displays the JIRA Backlog for the 'FP board'. It is organized into three main sections:

- FP1 Sprint 1** (2 issues): This section is expanded to show two issues:
 - 'Third story in my backlog' (FP-3, 3 points)
 - 'This is another story in my backlog' (FP-2, 1 point)
- Ready** (1 issue): This section shows one issue:
 - 'This is a story in my backlog' (FP-1, 5 points)
- Backlog** (3 issues): This section shows three issues:
 - 'The newest story' (My Test Epic, FP-5, 3 points)
 - 'The newest task' (My Test Epic, FP-6, 1 point)
 - 'The newest bug' (My Test Epic, FP-7, 1 point)

The next questions we have might be, *how do we know when we have enough story points loaded into our Sprint? How do we know when we have enough work in there?* And these are great question to ask.

Usually, we want to try and get a group commitment on our first Sprint, and then in our second Sprint we'll kind of do the same thing based on the output from our first Sprint. The number of story points that we complete in our Sprint is called our velocity. We use the velocity from our previous Sprint to help determine the velocity for the next Sprint. We're able to complete it and then adjust it accordingly. After three Sprints, we'll have what's called yesterday's weather, and the reason that they call it that is because yesterday's weather is known. We know what the weather was yesterday, but the weather tomorrow is a forecast, and that's the best we can do. It's usually pretty close, but it may not be exactly right. We can use yesterday's weather, which is the average of the velocity for the last three Sprints, and that becomes the maximum that we'll want to plan to. Given that, as a team, **FP Sprint 1** doesn't have a previous velocity, we're going to go ahead and commit to four story points, and we'll see whether or not we're able to complete them.

We've got our Sprint all set to go; the story points are in there, all of the stories meet our definition of ready, and we've even got some additional stuff to do in case we finish these things early. Let's go ahead and start our Sprint:


Start sprint

2 issues will be included in this sprint.


Sprint name:*

Duration:*

Start date:*

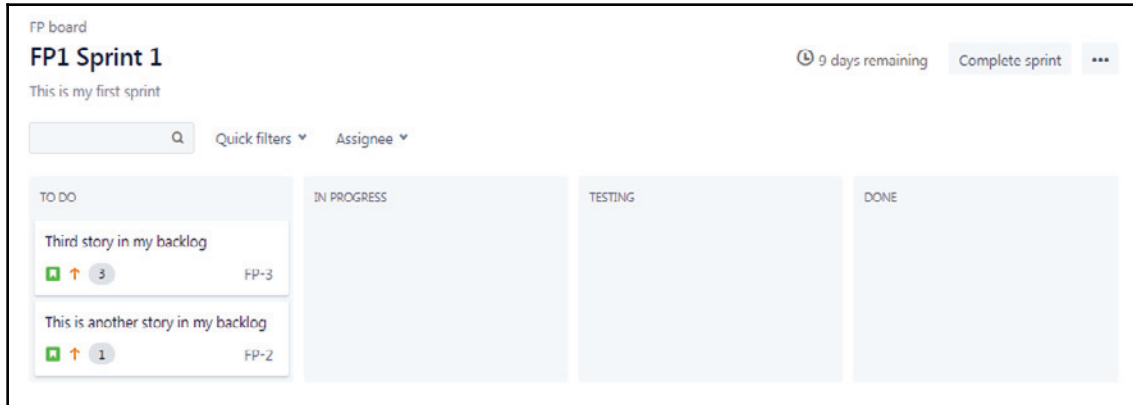
 

End date:*

Sprint goal:

We'll see that we have the ability to name the Sprint, which we've done already, the duration of this Sprint and its start and end dates, and the Sprint's goal. How many of these are working in the Sprint? Let's go ahead and look:



The daily Scrum

We've got a Sprint running. In this section, we're going to talk about the daily Scrum.

We are going to learn the following in this section:

- How to use JIRA daily as we're moving through a Sprint
- How to use the board and burndown points or hours
- How to know if Sprint is on track or off track

During a Sprint, we want to use JIRA, and we're going to try and use this tool so that we can move Sprint items through a workflow. We have our board with the columns on it—To Do, In Progress, and Done. We want to use JIRA to make sure that we're on track because if we know we're doing a two-week Sprint, we want to know on day two whether we're on track or not so that we can make adjustments in real time. We need to be able to generate data that we're going to need for reports at the end of the Sprint, and JIRA will do that for us automatically. We just need to follow some day-to-day process stuff. Then, we're going to facilitate the daily Scrum ceremony and help the Scrum master do a better job with their role.

Let's hop back over to JIRA. As you'll remember from the previous section, we created a Sprint called **FP Sprint 1**, we had our queue of ready work, and the rest of the backlog. However, since that Sprint just started, let's take a look at a Sprint that's already running. Let's hop over to our **Second Project**.

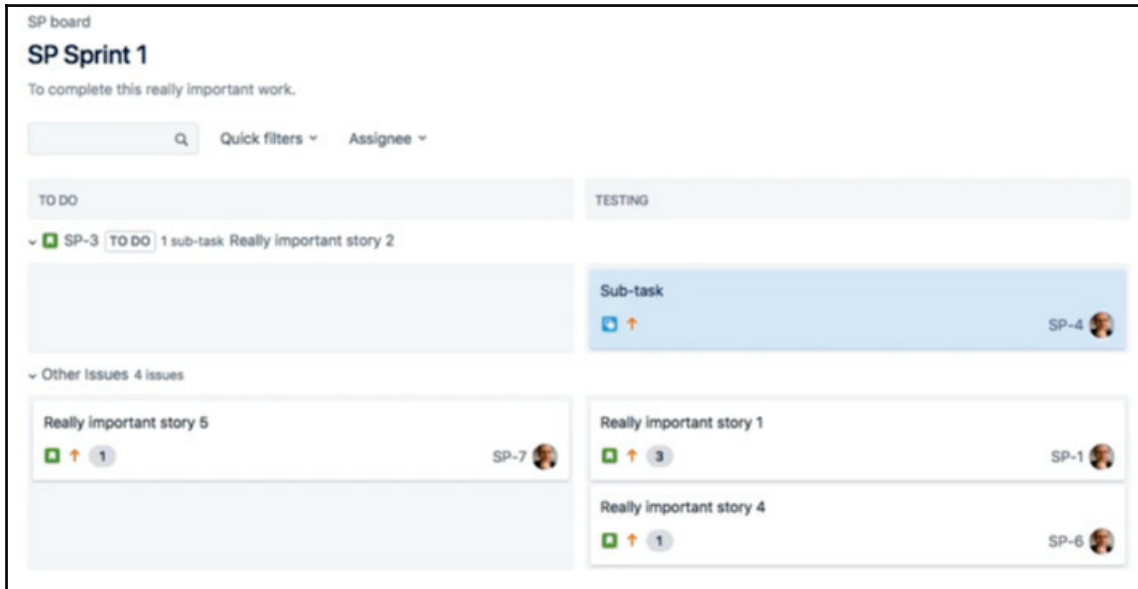
As we can see in the following screenshot, we are in the middle of **SP Sprint 1**:

The screenshot displays the JIRA Backlog for a project. It is organized into three main sections:

- SP Sprint 1**: Contains 5 issues, all labeled "Really important story 1" through "Really important story 5". Each issue is assigned to a sprint (SP-2 to SP-6) and has an upward arrow and a minus sign icon.
- Ready**: Contains 3 issues, all labeled "Really important story 6" through "Really important story 8". Each issue is assigned to a sprint (SP-7 to SP-9) and has an upward arrow and a minus sign icon. Below this section, it shows "+ Create issue" and "3 issues Estimate 0".
- Backlog**: Contains 2 issues, labeled "Really important story 9" and "Really important story 10". Each issue is assigned to a sprint (SP-10 and SP-11) and has an upward arrow and a minus sign icon. Below this section, it shows "+ Create issue".

At the top of the backlog, there are search and filter options: "Quick filters" and "Assignee". On the right side, there are buttons for "Plan sprint" and a menu icon (three dots).

We can see the items that are contained within the Sprint. There's the **Ready** Sprint of items that are ready to go, and then there's another couple of stories that are in the **Backlog** and priority order. When we are in a Sprint, we're going to want to look at the **Active sprints** view, which is really our board view:



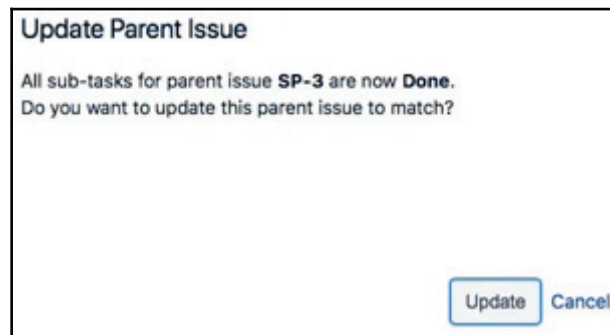
Active Sprints view

Imagine we're meeting in our daily Scrum and that we've brought our team together. We would be able to go around the room and ask people three questions that we want to ask at a daily Scrum:

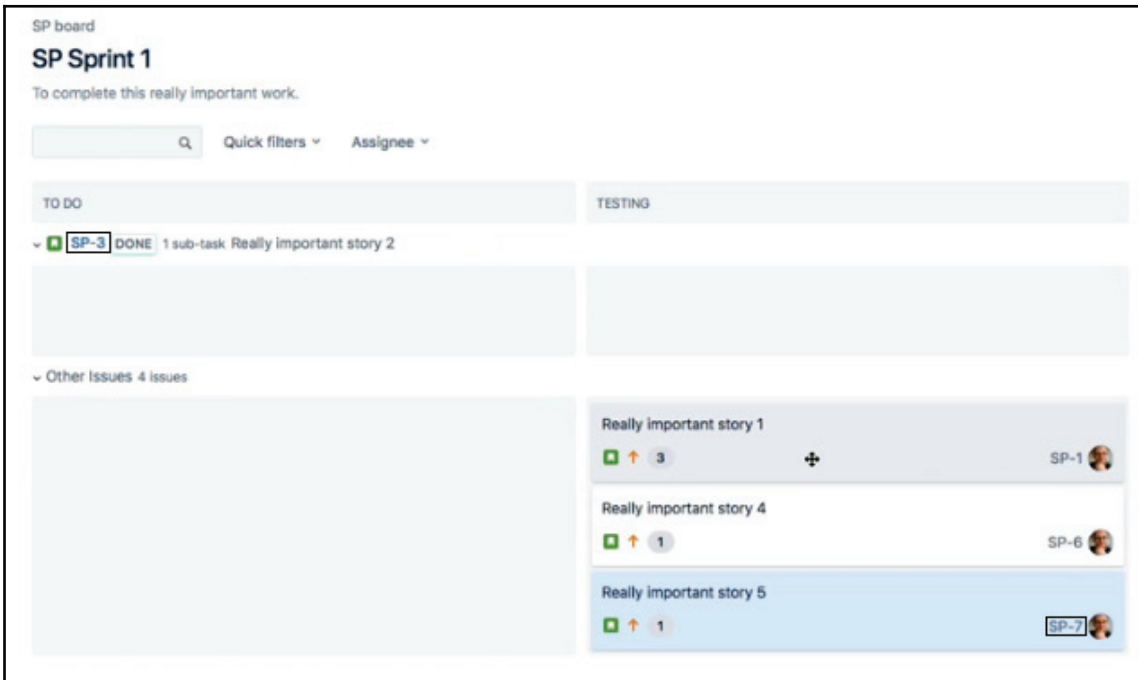
- What impediments do we have that are keeping us from meeting our Sprint commitment?
- Are we doing anything that will prevent our squad or other teams from meeting their Sprint commitment?
- Are there any new dependencies to discover or any ways to resolve dependencies that we have found out about since our last daily Scrum?

These are really the three questions we'll want to talk about with the team. As everyone is talking about the answers to those questions, and talking about what impediments they are running into that will help/will keep them from meeting their Sprint commitments, we'll be updating the board in real-time. They can do this prior to the meeting.

Let's take a look at the preceding screenshot. We've got a story, which is the **SP-3** story, which we can expand or collapse, and we also have a **Sub-task** that exists underneath it. We might have stories, and we might also have stories for subtasks. JIRA will display the smallest increment of work so that we can see that it's actually shown as the **Sub-task**. Below that, we can see our other issues, and these other issues are all story level items without subtasks underneath them. We can see that story number 3, or **SP-5**, has already been completed and moved to done. Because of this, we're going to move the **Sub-task** over to the **DONE** column, and we'll see that when we do that, JIRA will actually prompt us and say *all of the sub-task items under SP-3 are done, do we want to go ahead and update the status of the SP-3 item to done as well?* We can say yes:

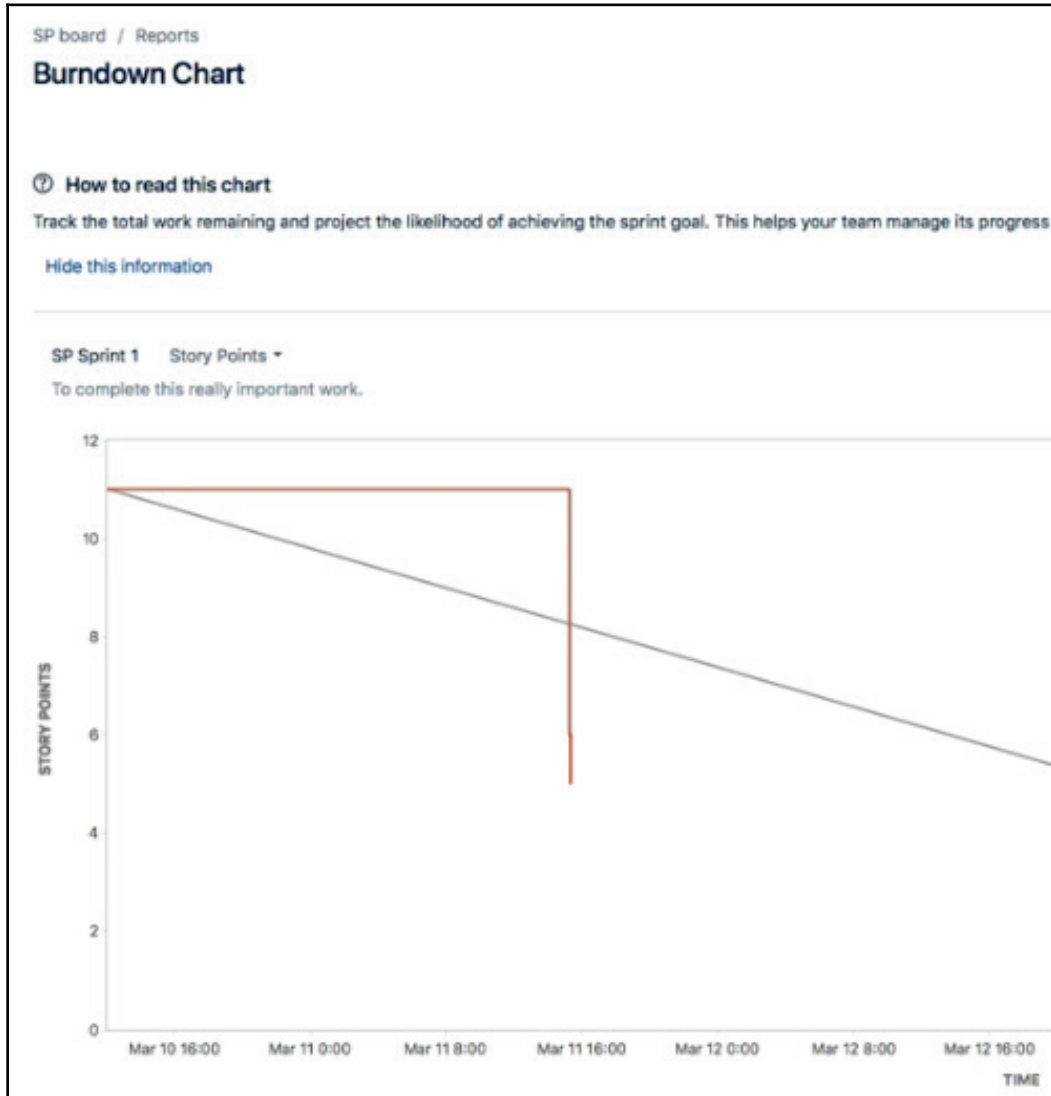


SP-3 is now complete, along with the sub-tasks underneath it. We'll move SP-7 to **in progress** because we have already started that one:



Active Sprints view

Let's take a look at our burndown and get a sense of whether or not we are on or off track for this Sprint. We'll go to **Reports** and look at the **Burndown Chart**. Let's take a look at how to read this **Burndown Chart**:



Burndown Chart

We can see on the left-hand side that we have **STORY POINTS**, and on the bottom we have **TIME**. What this shows us is that at the beginning of the Sprint, we had 11 story points, and we can see that as we carry on and we update them, we have less story points. Every time we move one of these items into the **DONE** column, that story's story points are decremented from the overall story points. We can see that the gray line in the preceding screenshot represents the ideal.

At the beginning of the Sprint, we had 11 story points, and at the end of the Sprint, we're going to have zero story points because they'll all be done. What this story point, or what this burndown, tells us is that for SP-1 when we're looking at story points (**SP Sprint 1**), we're actually ahead of schedule. We can see that we have a burndown of some of these story points and that five remained for the Sprint, and so we're actually ahead of the ideal line. We'll want to walk through this with our team. We'll get rid of the charts, including the burndown chart, and we can say, *hey team, looks like we're doing really well; we're actually ahead of schedule. There's a pretty good chance we're going to finish these items, in which case we'll want to be pulling them from the ready queue and adding more work.*

That's how we run a daily Scrum in JIRA.

Smaller stories or tasks

In this section, we are going to talk about whether we should have smaller stories or tasks, or how we should organize our work in the most effective way for our team.

In the previous sections, we set up our Sprint and started it, and we looked at how we should use JIRA within a daily Scrum. We'll talk about the work itself. We're going to talk about what is the best way for work to be structured in it. Should we have a story per person? Should we use subtasks under a story to make things more specific?

One of the things that makes this so challenging, and the reason that we ask this question repeatedly, is because really, it depends. This is one of those places where it depends on the team, it depends on how we work, and there's some key things we need to be thinking about as we're structuring our work. Really, it's about speed. It's about delivering the most work in the amount of time that a Sprint commitment is. Talking about the fastest teams and thinking about the concepts that help teams be fast helps us focus on the goals of the team and not the individual, and that's one of the things that's really challenging with this.

We can structure our work in a way that has stories per person, but really it's about making sure that the team is operating optimally as a team, not as a group of individuals. One of the effects of swarming on items to finish the most important work is that when everybody comes together and works on an item, that's where we see the most speed. Whether it's UX, development, backend development, and testers all attacking one story and moving through that one, and then attacking the next story, this is really the quickest way for the team to work as opposed to everybody working on their own thing. It helps to use fast performance so that we keep commitments realistic.

We previously talked about using yesterday's weather as a barometer for how much work to commit to in a Sprint. We can commit to more than that, and maybe we'll complete it, but it's really about team motivation. If we commit to 10 points and we complete 12, that makes the team feel happy; if they commit to 15 and they complete 12, the team feels like they didn't do a good job, and this will actually de-motivate them. We want to make sure that the team is motivated and excited about what they're doing. If we can keep the commitment realistic, this will allow the team to commit and finish the work that they committed to earlier, and then they can pull more from that ready cue that we created. This is really important because they're going to be delivering more than 100 percent of their commitment, and teams love that.

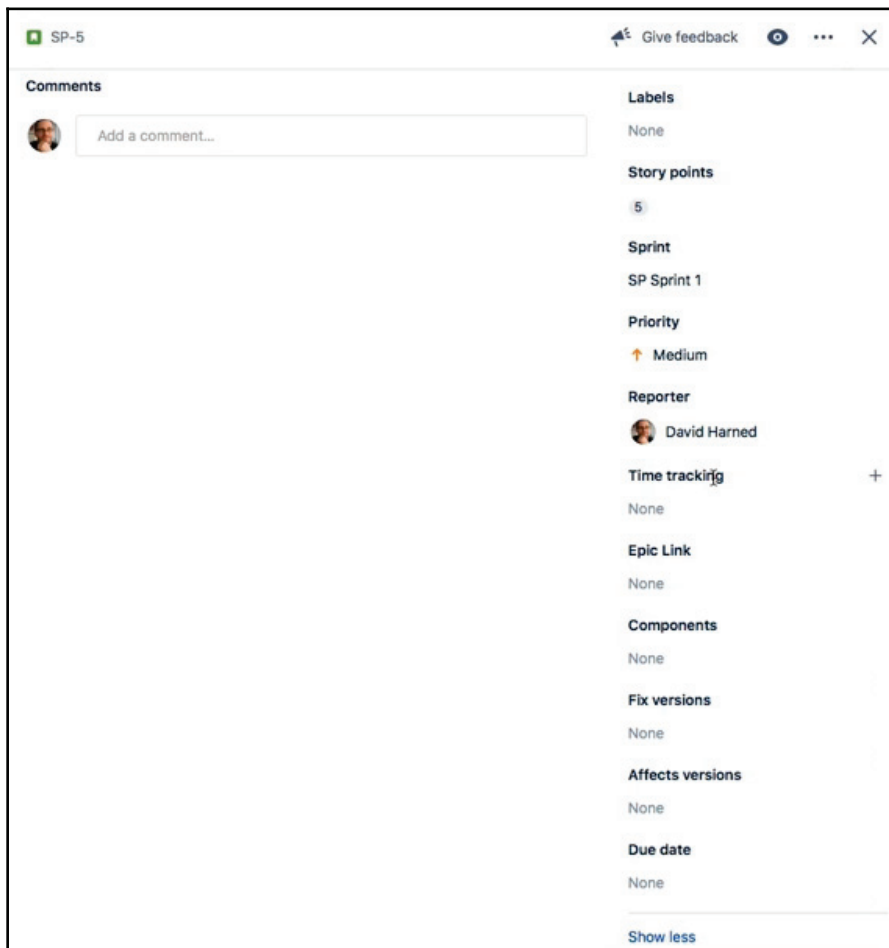
Next, we will look at relative sizing instead of specific sizing. If people commit to, *we're going to work through this story for eight hours, it's going to take me eight hours* and then it takes 10, using relative sizing allows us to not have to be on the hook for those specific time frames. If a story has three points, then we're able to relatively size that against the other stories in the Sprint, so a size three point story is the same as a three point story. We don't need to know whether that item is six and a half hours or eight and a half hours, we know it's a three.

The last concept is regarding having t-shaped people. This concept means that we can imagine a person with their arms outstretched to either side so that they look like a T. They're deep in a skill set in the middle of that T, and their arms represent two other skills that they might also have. The more we do to make our team cross functional, the more we can increase the throughput of our team. Maybe that tester is also pretty good at processing, and so they can serve us and help to serve as the Scrum master. Maybe we have a UX designer who can also do some frontend coding. This allows us to actually increase the speed of the project when people swarm on a story.

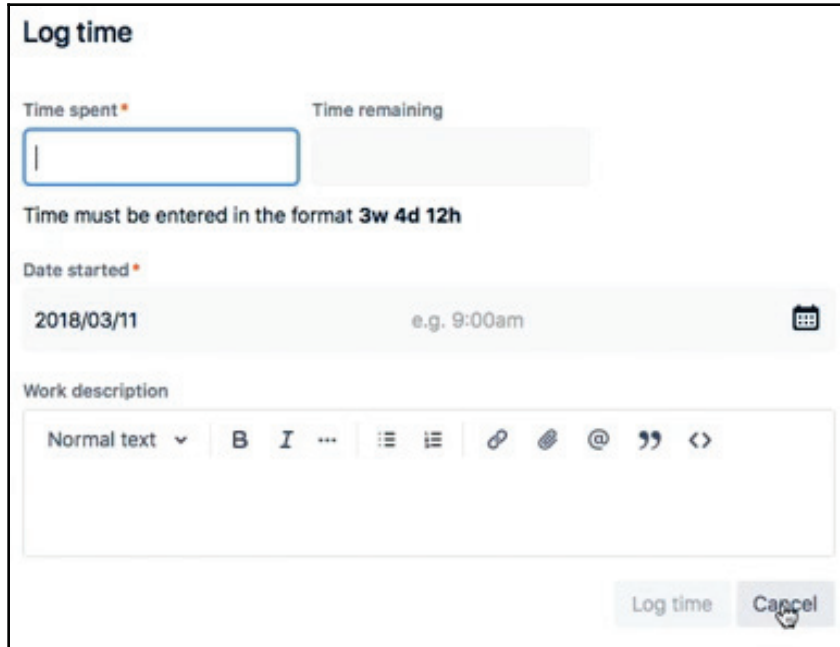
Let's take a look at JIRA and how JIRA would support those things. We'll talk through a couple of these concepts. Moving back into our **Backlog**, what's important to remember is that we talked about the fact that a board shows the smallest unit of work.

If we have a story in our backlog, we're going to work on this story. However, what we really want to do is create a task for UX, and we want to create a task for frontend development, and a task for backend development. We will have a person that will be on the hook for the story overall, and then we can create subtasks, which will allow me to capture each one of those tasks underneath. But again, remember, it's not about what's best for the individual, it's about what's best for the story and for the team overall.

We already talked about how when we move a story into done, we decrement those five story points from the Sprint, but if we want to track this in regard to hours, we can do that too. As we can see in the following screenshot, when we select any item, we receive a dialogue; under show more, we can see that we can actually do time tracking as well:



It's important to note that we can do this. The team should work in relative sizing. We can look at forecasting more effectively and things like that too, as some teams prefer to work in time:



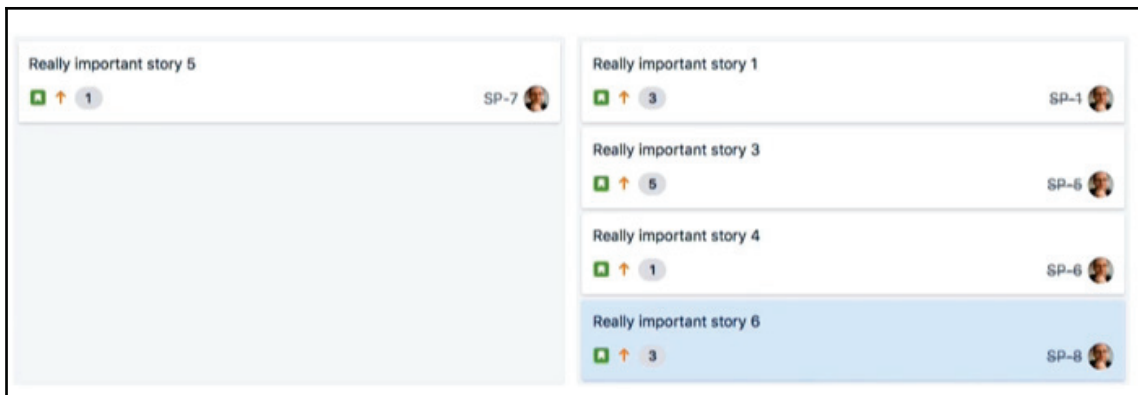
The screenshot shows the 'Log time' form in JIRA. It has a title 'Log time' at the top left. Below the title are two input fields: 'Time spent *' and 'Time remaining'. The 'Time spent *' field is currently empty and has a blue border. Below these fields is a note: 'Time must be entered in the format 3w 4d 12h'. Underneath is the 'Date started *' field, which contains the date '2018/03/11' and a time example 'e.g. 9:00am'. To the right of the date is a calendar icon. Below the date field is a 'Work description' field with a rich text editor toolbar. The toolbar includes a dropdown menu set to 'Normal text', bold (B), italic (I), a list of three dots, bulleted list, numbered list, link, unlink, email (@), quote, and code (<>) icons. At the bottom right of the form are two buttons: 'Log time' and 'Cancel'.

If we do, then what we can look at is the time spent and the time remaining, and then we can actually burn hours instead of story points in our burndown. On the left-hand side where we would have story points, we would instead have hours. If these were the number of hours that are required in order to finish our commitment, then those hours would burn to zero, which means that as we finish our story, we would want to update the time spent and the time remaining. Hopefully, this will help us figure out what's best for our team. Go ahead and experiment; this should be the way that we choose to handle our items in JIRA.

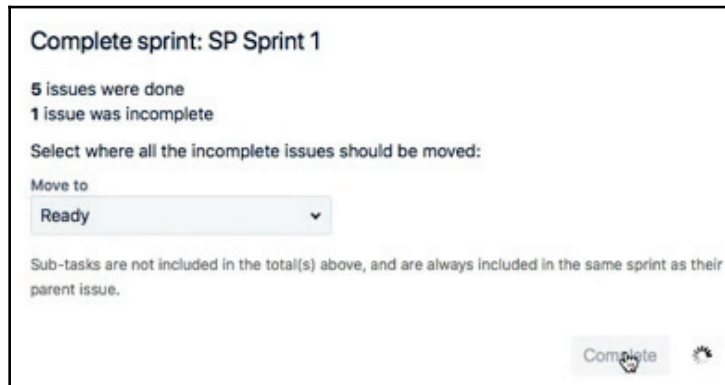
Closing the Sprint—the Sprint report

In this section, we're going to talk about closing our Sprint. In the previous sections, we talked about creating and starting the Sprint, we talked about the daily Scrum, how to use JIRA to run our daily Scrum and the value in doing that, and we talked about the way we should configure our work and whether we should use smaller stories. Should we use tasks? How do we break it down to a point where it's most useful for the team? In this section, we will talk about closing the Sprint and learn how to complete a Sprint.

Let's take a look at JIRA. We were working on **SP Sprint 1**. We've finished **SP-3** and the subtasks underneath it. We've finished really important stories number **1**, **3**, **4**, and **6**, and **3** is up, a really important story number **1** as shown in the following screenshot. We can see that these are all done. However, story number **5** is not done, which means that we weren't able to finish it. This being said, we were able to get story number **6**, pull it into the Sprint, and were able to complete it. We have four days remaining on this Sprint, as shown in the following screenshot, but we can still go ahead and complete our Sprint:



Next, we're going to hit the **Complete** button, and from the following screenshot, we can see that five issues were completed and one issue was incomplete. It prompts me and says, *where do we want to put that incomplete issue? Do we want to put it in the Backlog, or do we want to put it in the Ready Sprint?* We'll go ahead and put it back in the **Ready** Sprint so that we can move the incomplete item back into ready and complete our Sprint:



Complete sprint: SP Sprint 1

5 issues were done
1 issue was incomplete

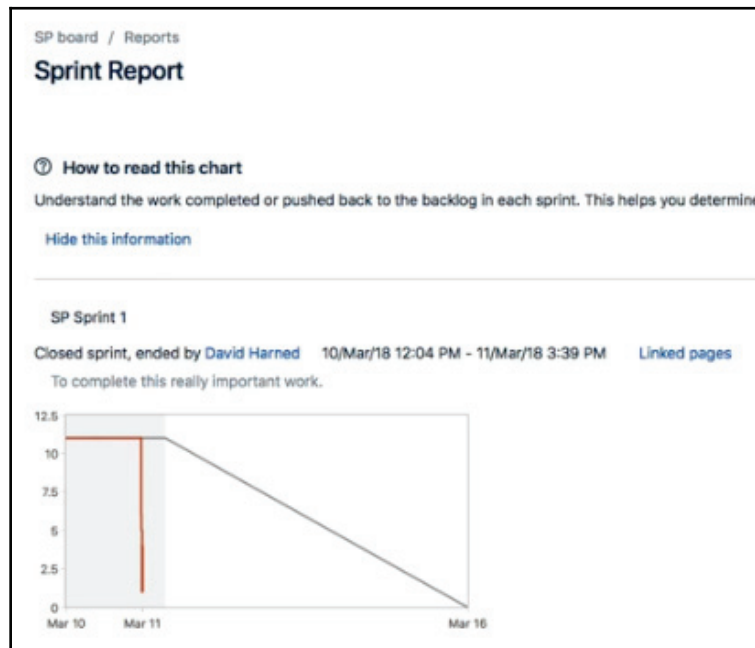
Select where all the incomplete issues should be moved:

Move to
Ready

Sub-tasks are not included in the total(s) above, and are always included in the same sprint as their parent issue.

Complete

What we're seeing in the following screenshot is called a **Sprint Report** for this Sprint:



Sprint Report

There's a few things to look at. First of all, we talked about this burndown, and we can see in the preceding screenshot that we originally started with 11 story points, and we were able to burn down to one. We completed 10 story points in this Sprint. We can see the completed issues and we can also see the items, the number of those items, and the amount of story points that were done.

Then, there's **SP-8**, which is story number six as shown in the following screenshot, that we added after the Sprint had started. We can see the issue that was added to the Sprint after its start time:

Completed Issues		
Key	Summary	Issue Type
SP-1	Really important story 1	Story
SP-3	Really important story 2	Story
SP-5	Really important story 3	Story
SP-6	Really important story 4	Story
SP-8 *	Really important story 6	Story

We did not complete issue number seven, **SP-7**, which was really important story number five, so that was one story point that we didn't get.

Issues Not Completed		
Key	Summary	Issue Type
SP-7	Really important story 5	Story

Issues not completed

The good news is that we were able to complete 13 story points on our 11 point commitment and the bad news of course is that we didn't finish the one story point. We did complete 13, we did have 11 as our commitment, and we did deliver our commitment plus a little more. Hopefully, not completing this one story was okay for our product owner.

Summary

In this chapter, we learned how to use JIRA to run our project, how to create a Sprint, and how to bring refined backlog items in so that we can use a ready Sprint that meets the definition of ready. We determined how big the right size for a Sprint commitment is using yesterday's weather. We also learned how to start a Sprint, how to use JIRA during a daily Scrum and throughout the Sprint, and we looked at burndowns and board views to do that.

We talked about how to best help the Sprint succeed using JIRA as a tool, different concepts to structure work so that our team is most effective as they're moving through the Sprint in the board view, either through subtasks or smaller stories, and then finally we learned how to complete a Sprint.

In the next chapter, we're going to talk about reporting.

4 Working with Reports

In this chapter we are going to learn about versions and releases—what they are and how they're different from each other. We'll talk about how to read burndowns, Sprint reports, and velocity charts to determine whether or not your team is doing well. We will take a look at releasing epic burndowns as well as versions and epic reports, which give us the ability to do forecasting, which is very powerful.

This chapter will cover the following topics:

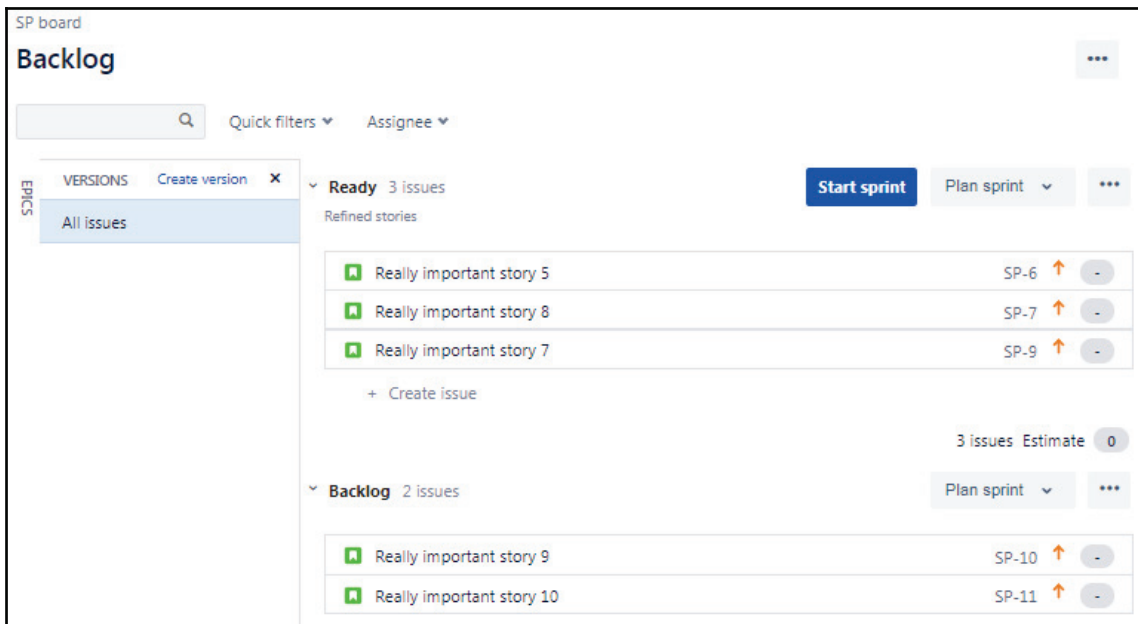
- Versions and releases
- Burndown report
- Sprint report
- Velocity chart
- Release and epic burndowns
- Version and epic reports

Versions and releases

Let's start with versions and releases. In this section, we're going to learn about what versions and releases are and how they interact with one another. We're going to talk about how to create and manage these versions, how to assign work to two versions, how to perform a release, and then finally how to view the contents of a release once we've made one.

When we talk about versions and releases, versions come from the concept of software. We're all familiar with major and minor versions of a software, for example, Version 2.0 versus Version 2.1. They really represent a large amount of value, so that's a way to think about it: a version represents a chunk of value that's being deployed or released. Versions can be released, so once we've determined what's in a version, and we complete that version, then we can release it, and then that version becomes a release. They're the same thing—just one is pre-deployment and one is post-deployment. Once a version has been released, we can view the contents of that version as release notes, so we'll take a look at that too.

Let's hop over to JIRA and take a look at our **Second Project**. In the **Backlog**, we can see that we have **VERSIONS**. We can actually have a look at these versions by clicking on them:



The screenshot shows the JIRA Backlog for a project named 'SP board'. The interface includes a search bar, quick filters, and an assignee dropdown. A sidebar on the left shows 'VERSIONS' and 'All issues'. The main area is divided into two sections: 'Ready' (3 issues) and 'Backlog' (2 issues). The 'Ready' section contains three items: 'Really important story 5' (SP-6), 'Really important story 8' (SP-7), and 'Really important story 7' (SP-9). The 'Backlog' section contains two items: 'Really important story 9' (SP-10) and 'Really important story 10' (SP-11). Each item has an upward arrow and a minus sign. A 'Start sprint' button is visible in the 'Ready' section, and a 'Plan sprint' button is visible in the 'Backlog' section. The '3 issues Estimate' is shown as 0.

As we can see in the preceding screenshot, this is where all our bot versions will be stored. We mentioned before that a version is a release before it's been released; they're the same thing. Let's go under **Releases** and have a look at the following screenshot:

Create version

Name*
test version

Start Date Release date
e.g. 2018/12/31 e.g. 2018/12/31

Description
test version

Save Cancel

We can see in the preceding screenshot that we have the version **Name**, a **Start date**, the **Release date**, and a **Description**. We will make a version and we'll call it `test version`. Our **Start date** and **Release date** are optional, but let's go ahead and write `test version` in the description, as shown in the following screenshot. Then, we'll click on **Add**:

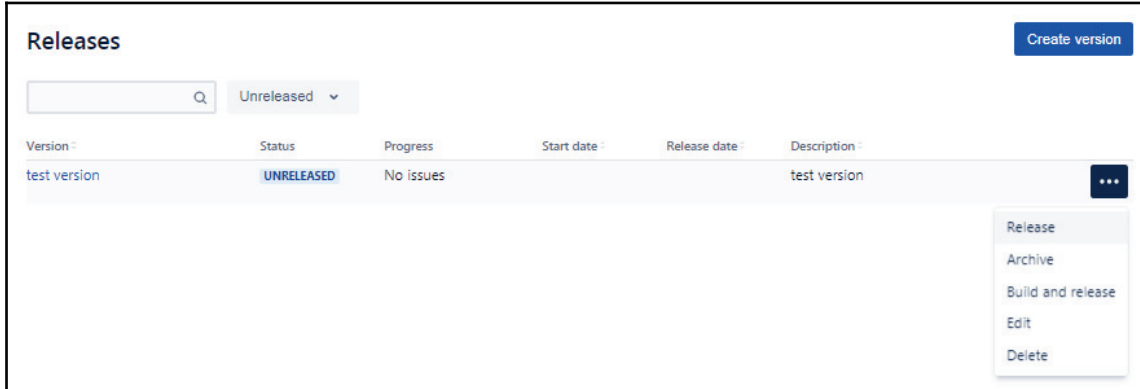
Releases Create version

Unreleased

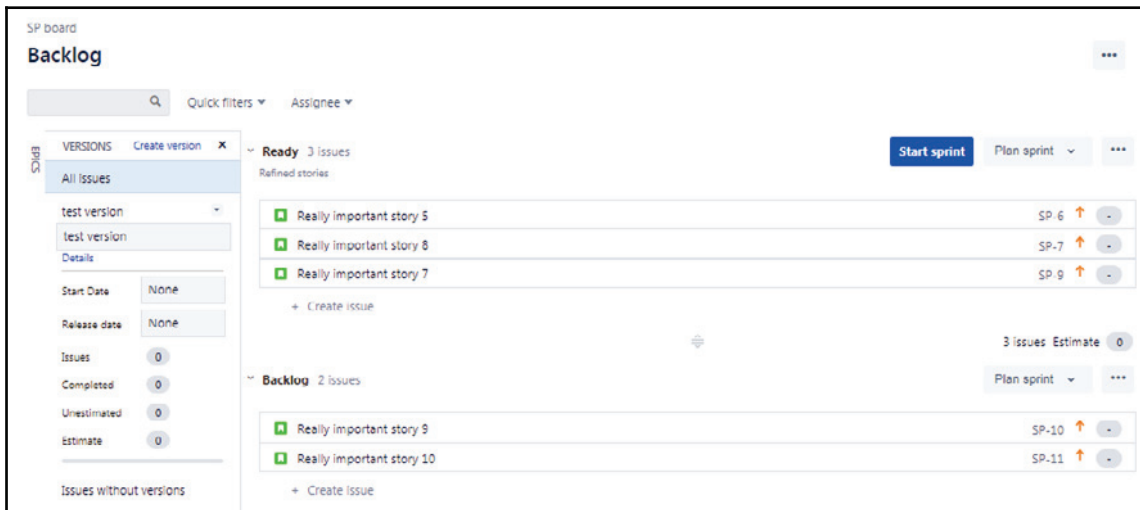
Version	Progress	Start date	Release date	Description
test version	No issues			test version

- Released
- Unreleased
- Archived

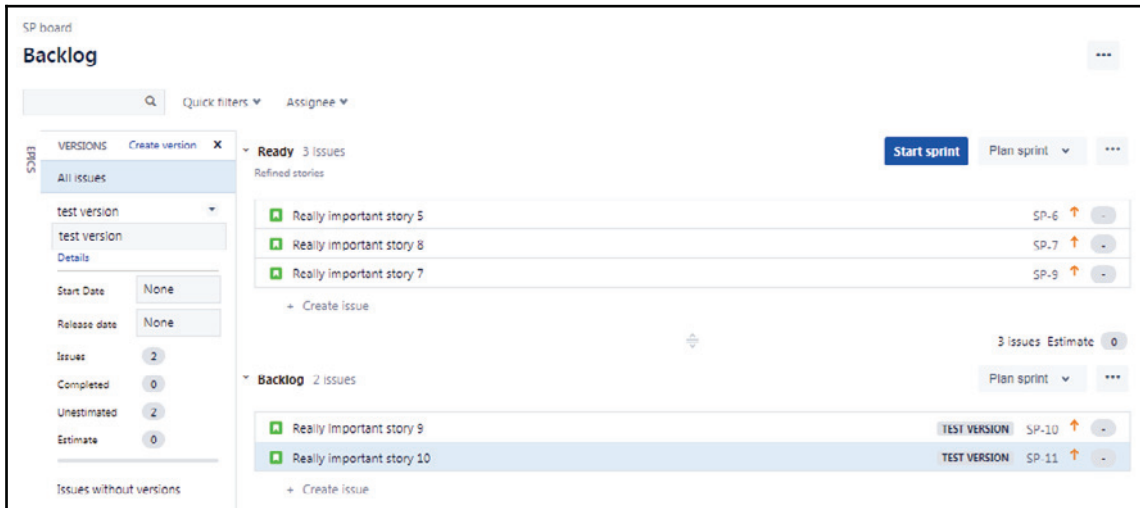
We have the option to look at **Released**, **Unreleased**, and **Archived** versions. We also have our **test version**. We can create another one if we like. In the following screenshot, we can see that we have the ability to **Release**, **Archive**, and more:



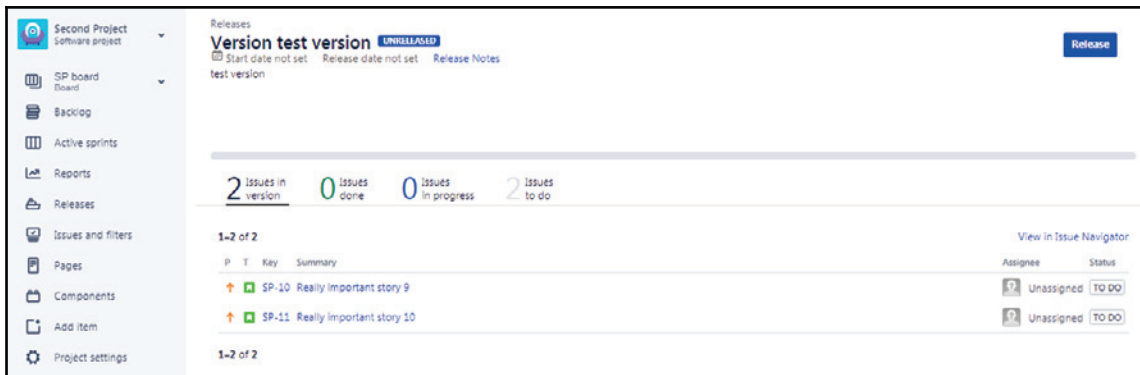
If we go back to our **Backlog**, we can see that if we click on the **VERSIONS** tab on the left, as we can see in the following screenshot, we actually have our **test version**. We can see the number of **Issues**, **Completed**, **Unestimated**, and more:



We can actually take two stories and drag them in. If we do that, we can see in the following screenshot that these stories are now a part of the **test version**:



If we go under **Releases** again, we can take a look at the version. We can see in the following screenshot that there are 2 issues in this version, the two that we added, and there are 0 in Issues Done, 0 in In Progress, and 2 issues in To Do. Once we're done with this, we can actually hit **Release** and release this item:



We will go back to the **Backlog** and we'll put the other three stories into **VERSIONS**. This means that we now have five issues. We'll go back to **Releases** and then we'll go ahead and release our version:

The screenshot shows the Jira Releases interface for a version named "Version test version" which is currently "UNRELEASED". At the top, there are fields for "Start date not set", "Release date not set", and "Release Notes", along with a "Release" button. Below this, a summary bar indicates: 2 Issues in version, 0 Issues done, 0 Issues in progress, and 2 Issues to do. A table below shows the first two issues:

ID	Key	Summary	Assignee	Status
SP-10	Really important story 9		Unassigned	TO DO
SP-11	Really important story 10		Unassigned	TO DO

As we can see in the preceding screenshot, we've released the items, we can see the item in progress, and we can see that some are in the to do list. If we go to our **Release notes**, then we can see what was contained within our version:

The screenshot shows the "Release notes" page for the "test version". It features a "Story" section with a bulleted list of items included in the release:

- [SP-7] - Really important story 5
- [SP-9] - Really important story 7
- [SP-10] - Really important story 8
- [SP-11] - Really important story 9
- [SP-12] - Really important story 10

Below the list, there is a text area for copying the release notes to another document. The content of the text area is as follows:

```

Release notes - Second Project - Version test version
<h2> Story
</h2>
<ul>
<li><a href="https://digitalcoffee.atlassian.net/browse/SP-7">SP-7</a> - Really important story 5
</li>
<li><a href="https://digitalcoffee.atlassian.net/browse/SP-9">SP-9</a> - Really important story 7
</li>
<li><a href="https://digitalcoffee.atlassian.net/browse/SP-10">SP-10</a> - Really important story 8
</li>
<li><a href="https://digitalcoffee.atlassian.net/browse/SP-11">SP-11</a> - Really important story 9
</li>
<li><a href="https://digitalcoffee.atlassian.net/browse/SP-12">SP-12</a> - Really important story 10
</li>
</ul>

```

Release notes

Burndown report

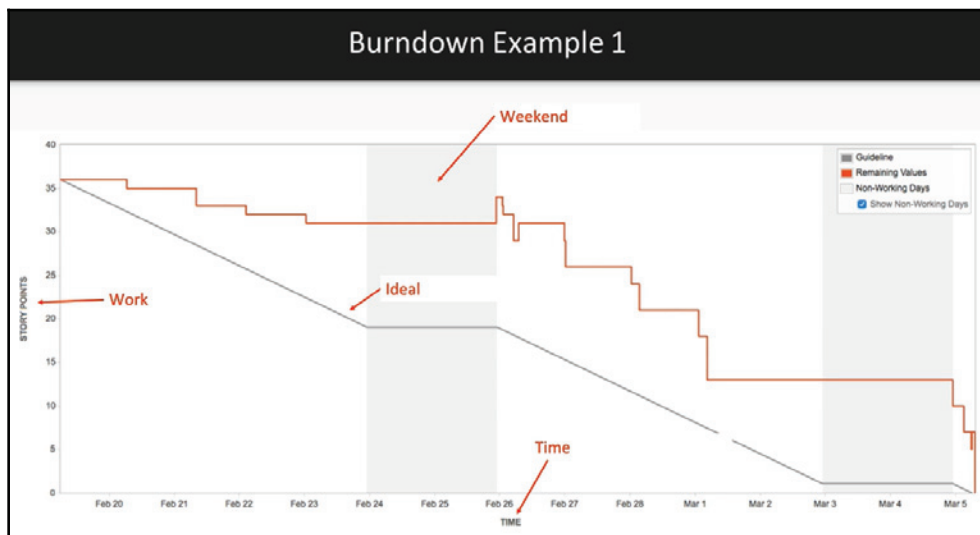
We're going to talk about the burndown report. We've spent a little time previously looking at the burndown report in regards to how we use it during running a Sprint iteration, and how we use it with our team. But in this section, we're going to talk more specifically about the report itself and how we read it, and then what kinds of data we can use for a burndown report.

As we already know, burndown is used to measure progress within an iteration and helps us understand whether we are on track or off track from our ideal state. In a burndown, the vertical axis represents the total amount of work that exists inside of that iteration, and the horizontal axis represents time.

Burndowns tell stories. The more we look at them, the more we understand what may or may not have happened during an iteration, and we get pretty good at telling the story from looking at those burndowns. In a burndown, we can burn down all kinds of things, so we can burn down points, we can burn down hours, we can burn down risks, and there's other things too.

Burndown example 1

We'll now take a look at how to set these things up in JIRA. Let's look at this first example:

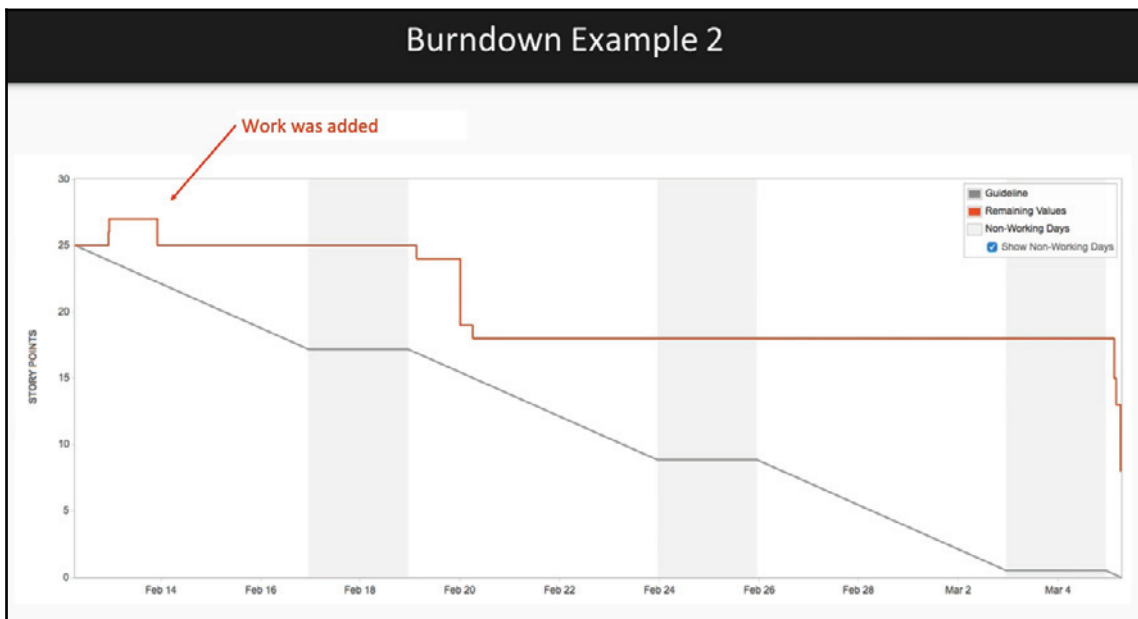


Burndown example 1

First of all, we can see on the vertical axis that we have **STORY POINTS**. In this iteration, we've got about 36 story points at the beginning of this Sprint. We can see at the end that we actually want to burn down two 0 story points, so ideally we're handling all of the work that was committed to in this iteration. On the bottom, we can see that we have **TIME**. As we can see, this would be a two-week Sprint, and we can see that as we go through time, the ideas that we burn down makes the story points go down to zero. The gray line represents the **Ideal** scenario, so, assuming that the work is being sorted through correctly, we want to be close to that gray line, as this would show the pattern that we're looking for. The gray bars in the preceding graph, on the other hand, represent a **Weekend**, or a time that the team is not working, and so we can see that they're flat because ideally we're not working on the weekends. We can also see that there's a blue check box in the upper right-hand side that says **Show Non-Working Days**. We can uncheck that and get rid of those weekends if we don't want to look at them.

Burndown example 2

We'll be talking about how burndowns can tell us a story about the Sprint. Let's take a look at the following Burndown example:



Burndown example 2

One of the things that we want to keep an eye on, as we can see at the beginning of this particular Sprint, is that we started with 25 story points, and we didn't quite burndown to zero. As we can see in the preceding screenshot, there's a little hump in red, and this hump is caused by work being added to the Sprint once it's already been started.

Burndown report

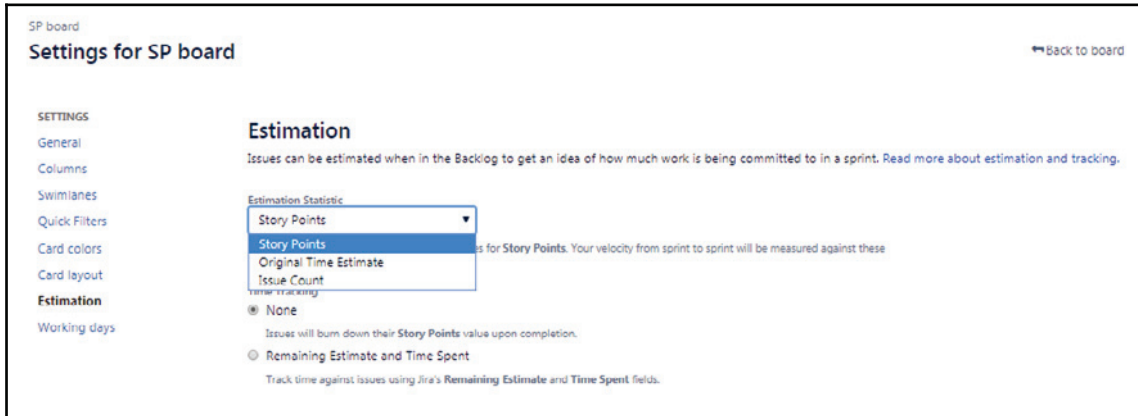
Let's take a look at JIRA—we want to have a look at a couple of things. First of all, the way to get to a burndown is to go to **Reports**. **Burndown Chart** is the first item that we'll see, and this is where we can get to our burndown:

The screenshot shows the JIRA Reports interface for a project named 'Second Project'. The left sidebar lists various reports under the 'AGILE' section, including Burndown Chart, Burnup Chart, Sprint Report, Velocity Chart, Cumulative Flow Diagram, Version Report, Epic Report, Control Chart, Epic Burndown, Release Burndown, and Issue Analysis. The main content area is titled 'All reports' and 'Agile', displaying a grid of report thumbnails with their respective titles and descriptions:

- Burndown Chart**: Track the total work remaining and project the likelihood of achieving the sprint goal. This helps your team manage its progress and respond accordingly.
- Burnup Chart**: Track the total scope independently from the total work done. This helps your team manage its progress and better understand the effect of scope change.
- Sprint Report**: Understand the work completed or pushed back to the backlog in each sprint. This helps you determine if your team is overcommitting or if there is excessive scope creep.
- Velocity Chart**: Track the amount of work completed from sprint to sprint. This helps you determine your team's velocity and estimate the work your team can realistically achieve in future sprints.
- Cumulative Flow Diagram**: Shows the statuses of issues over time. This helps you identify potential bottlenecks that need to be investigated.
- Version Report**: Track the projected release date for a version. This helps you monitor whether the version will release on time, so you can take action if work is falling behind.
- Epic Report**: Understand the progress towards completing an epic over time. This helps you manage your team's progress by tracking the remaining incomplete/unestimated work.
- Control Chart**: Shows the cycle time for your product, version or sprints. This helps you identify whether data from the current process can be used to determine future performance.

All reports

The other thing we want to talk a little bit about is burning down other values. If we go back to our **Backlog** in the upper right corner, we can see that we have our **Board settings**, so we'll click that, and then under our **SETTINGS**, we've got **Estimation**:

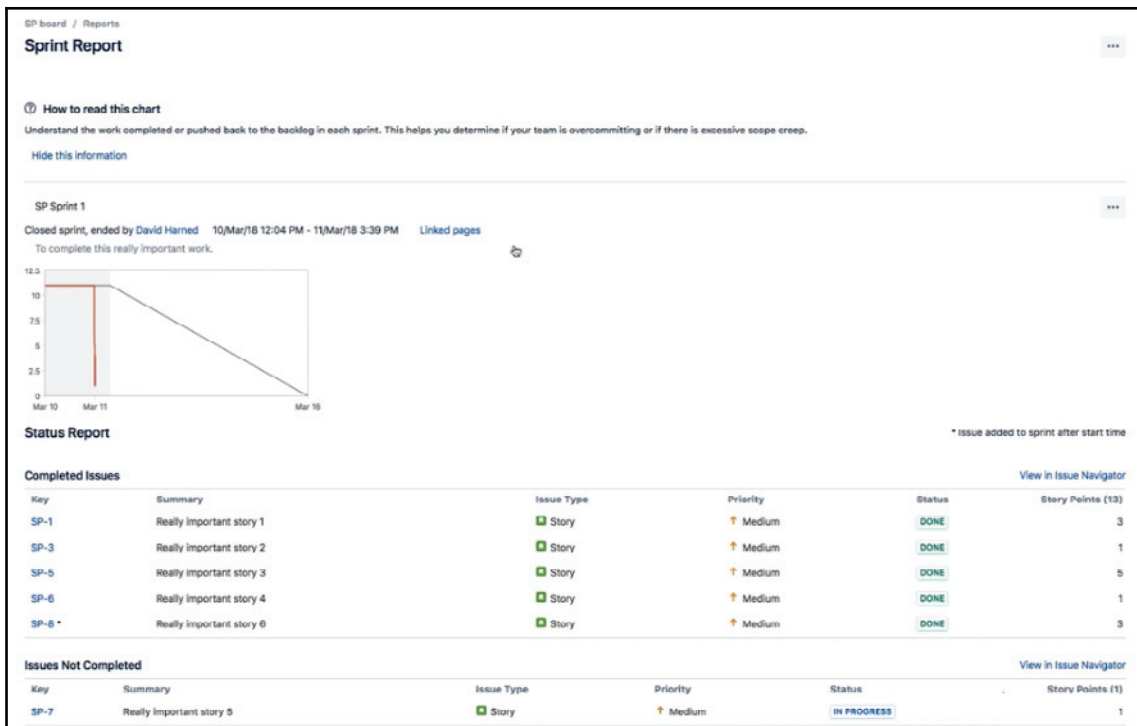


And we can see in the preceding screenshot, we're estimating in story points, but we've also got the ability to estimate in original time as well. Then, we can use this time tracking so that instead of seeing story points on that vertical axis, we'll actually see hours, and so we can burndown hours. As we move through the Sprint items, let's say we've got an item that started with 10 hours, we can then update that value to be, say, 8 hours or 6 hours long, and it will actually burn down the amount of hours as we move through the Sprint.

Sprint report

In this section, we're going to talk about the Sprint report. We'll talk about what it is, and we'll talk about how to read it. In the Sprint report, there's a summary of the Sprint iteration. It's going to show us the burndown, the work that's been completed, the work that wasn't completed, and any work that was added and removed during the iteration.

Let's take a look at JIRA, and get more information about a Sprint report. Do we remember the demo project that we used? For going into reports, we can click on **Reports** on the left, and then we can take a look at our Sprint report:



Sprint report

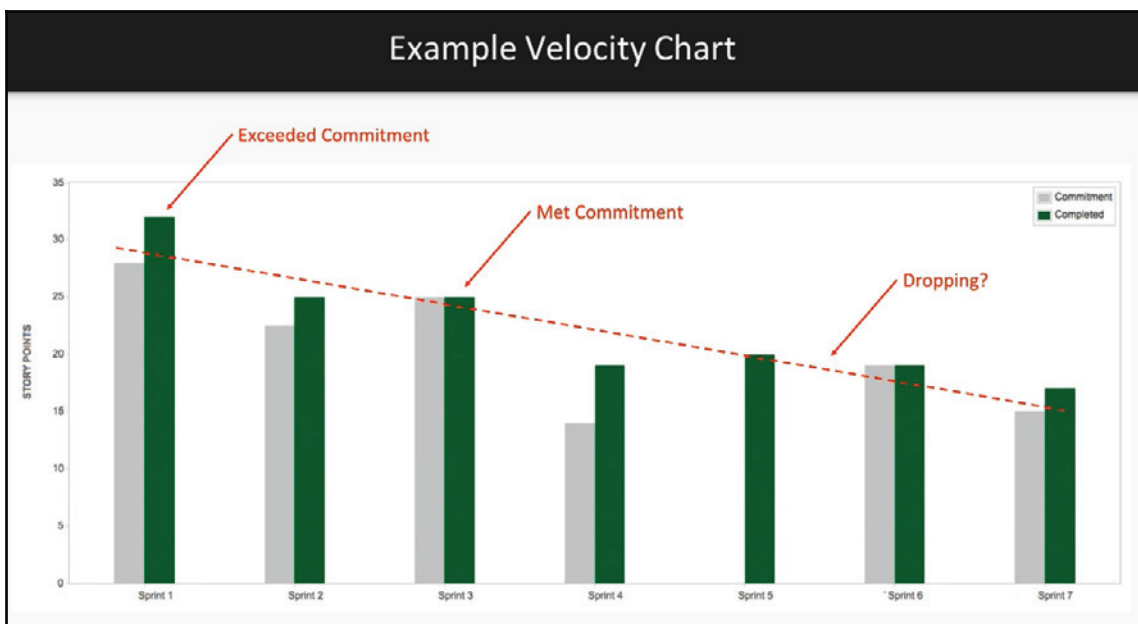
Keep in mind that our burndown is not going to be beautiful because this was a one-day Sprint. Normally, the burndown would be more appropriate based on the iteration length of our we Sprint. But let's pay attention to the bottom. As we can see in the preceding screenshot, we've got the **Completed Issues** (items that were completed during this Sprint). If we look at any issues that were added after the Sprint started, we'll notice that they have an asterisk by them, which allows us to see any items that came in that were not part of the original commitment. We can also see items that were not completed, so **SP-7** was not completed during this Sprint. We can also see any items that were removed from the Sprint. In total, we've got completed items, we've got added items, we've got removed items, and then anything not completed. Of course, we also have a Sprint name and we've got the date range for that Sprint as well.

Velocity chart

In this section, we're going to talk about the velocity chart. The velocity chart is very valuable. First, we'll talk about what it is, how to read it, and then we're going to talk about how to use our past velocity to plan our future commitments.

Velocity chart example

Let's take a look at an example of a velocity chart:



Velocity chart

To orient us to the preceding chart, let's take a look at the different elements. The gray bars represent what we've committed to in our Sprint, and we can see that this is measured in **Story Points**. In this example, we can see that in **Sprint 3**, we've actually committed to **25** story points. The green bar represents what people created, and so we've completed **25** story points, and we've met **100** percent of our commitment. That's great! We can also see some examples where the gray bar is lower than the green bar, which means we've committed, but we've actually completed even more than we committed, which is even better.

If we look at **Sprint 5**, it doesn't have a commitment, so there may have been some sort of data problem there, or there was no commitment and the Sprint was started without a commitment in place, which means we would probably see a burn up when Sprint work was added after the beginning of the iteration. We have already talked about reports telling a story; this report tells a different story, which is interesting. We can see that our velocity is actually dropping, which would definitely be a cause for alarm and it'd be something to talk about with our Scrum master or with the team if we're looking at this report. It's important to get some information about what's going on and use this to gather some knowledge about what's going on with this project. That was the velocity chart.

Release and epic burndowns

In this section, we're going to talk about release and epic burndowns, how important they are, and how we can use them to give us some insight into how things are going. We're going to learn about what release and epic burndowns are, how to read these burndown reports, and how to use these reports for forecasting.

Release burndown example

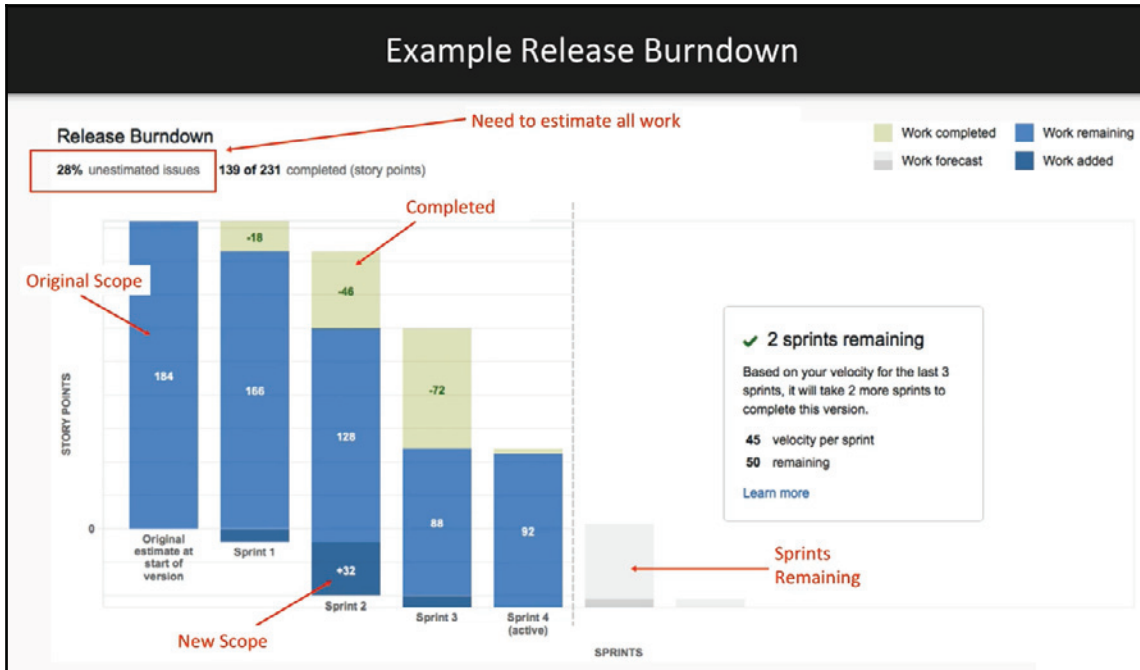
We're going to show an example of a release burndown.



Keep in mind that it releases a version. Essentially, it's a version that's been deployed. An epic is also a container of work.

Recall to our backlog. A release is generally larger than an epic, so an epic would be a large story that spans multiple Sprints, but the same concepts apply, and so when we're looking at an epic burndown report, it's going to be the same as a release burndown report, but probably smaller. However it will allow us to project when that epic might be completed.

Let's take a look at this release burndown. We want to familiarize it with the different elements of this report:



Release burndown example

First of all, what we're seeing in the first column is our estimate of the work that's contained within this version, which will eventually be our release. We can see that it has **184** points. The thing that's interesting about this report, in this specific example, is that we have **28%** of the work that's unestimated, which means that it's hidden. There's **20%** of the issues that are contained within that do not have story point estimates, which means that our original scope is actually larger than the **184**, but we don't have that work sized, which means that that would be very important to go and make sure that all of the work is sized so that it's all contained within our forecast right away. As we said before, those **184** points represent our original scope, and so that medium blue bar represents the original scope of the version at the time of creation. The dark blue in the preceding example, plus **32** points from **Sprint 2**, represents the new scope. This is work that was added in this iteration to this version and is now part of this release. We can see that the green bar represents work that's been completed, so in **Sprint 2**, we finished **46** points of the original scope and added **32** points. We still have **128** points of the original scope remaining. Given this, we can see that we are starting to establish a velocity. We know that this team's velocity is **45** points per Sprint, and, based on that, we can see the dashed line and where we are today. We know that we have **50** story points remaining in the scope of this version. This tells us that we've got just over a Sprint worth of work, so essentially two Sprints remains. If our two Sprints are two weeks each, then we can see that we have essentially one month remaining before this version will be available for release, and that gives us the ability to forecast.

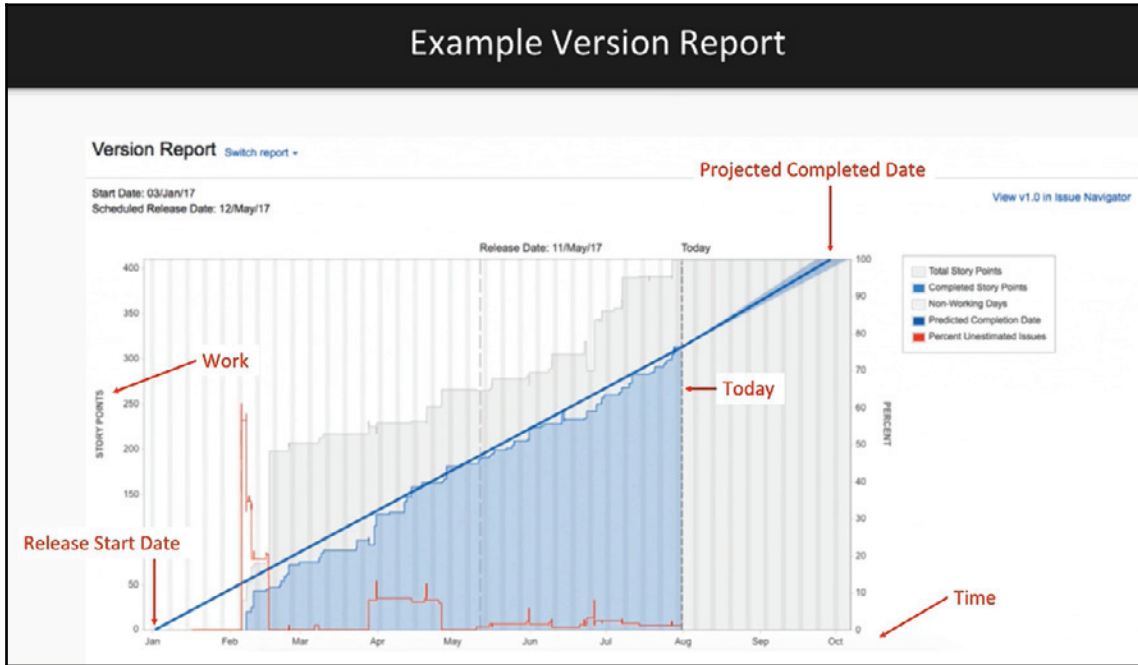
Version and epic reports

In this section about reporting, we're going to talk about version and epic reports. In the previous section, we talked about release and epic burndowns, and we're going to get a similar kind of view in this report, but it's a different look, with slightly different information. In this section, we're going to learn about what version and epic reports are. Again, they're very similar.

A version report is large and contains a larger piece of value and an epic is small. It is basically the same kind of report, but it allows us to look at a specific version or a specific epic, and then we get some sort of forecast as to when that item will be completed. We'll look at how to read a version report, and then we'll also look at how we can use these reports to provide a forecast.

Version report example

We're going to look at an example version report, as follows:



Version report

As we can see in the preceding figure, we've got the work in the vertical axis represented in **STORY POINTS**. We can see that there's actually quite a few story points in the preceding report, because usually a version has a lot of functionality built into it. We can also see that the horizontal axis represents **TIME**. There's a red line in the report which represents the amount of unestimated work as it goes up and down, as we move through this version. We can see from the blue line that this version started in January, we can see **Today**, which is represented by a dotted line, and we can see that we have a projected completion date. Some other things to note is the completed date is a cone shape, and that there's an optimistic date a little bit to the left of that projected completion date, and a little bit to the right of that is a pessimistic date.

We actually have an optimistic and pessimistic release date and projected based on the actual velocity of the team and the work that's included inside of that version. This value is actually dynamically generated as we move through time. If we remove the scope from this version, this will pull the date in and it will actually deliver earlier, and if the team slows down or if we add more scope, then that date will get pushed out. The closer we get to the actual release date (the projected completion date), the tighter that cone gets until there's no more cone left at all. When we first started in February or March, we'd have a much broader wider cone, given the fact that there's a lot of variables that could affect that completed date. This gives us a useful ability to look at this version over time, and figure out what that completion date might look like, and do some things very far ahead that would allow us to affect that. Should we bring the team together into one room, should we add more people, more to resources to the project, should we change the scope, and so on, we've got lots of choices assuming we've got a good view of what's coming and what lies ahead.

Summary

That brings us to the end of this chapter. In this chapter, we learned about what versions and releases are, what iteration release epic burndown reports have and how we should read them, what a Sprint report is, the data that's inside a Sprint report and how we can read it, what a velocity report looks like and why it's important for planning future Sprints, what version and epic reports are and how to read them, and then we also talked about version and epic reports and used those to provide forecasts for when our version will be completed.

In the next chapter, we're going to talk about searching and filtering.

5

Searching and Filtering on Issues

In this chapter, we're going to be talking about searching and filtering on issues, which can be a very powerful capability within JIRA.

We are also going to talk about JQL, what it is, how to write queries in JIRA using simple and advanced editors, and how to export WER results.

In this chapter, we will be covering the following topics:

- Issue searching using JQL
- Saving and managing filters
- Executing bulk changes
- Creating new boards from saved filters

Issue searching using JQL

In this section, let's talk about issue searching using JQL. We are going to talk about what JQL is, writing queries in both the simple editor and the advanced editor in order to return results, and how to export those results to use them in other ways.

First, let's talk about JQL. We don't want to confuse this with the Java query language, which is something different—it's the **JIRA Query Language** that we want to look at. It's very similar in format to SQL, so if we've spent any time in SQL and understand that query syntax, we're going to feel pretty comfortable in JQL. It uses fields, values, operators, and keywords. Let's talk about what those are.

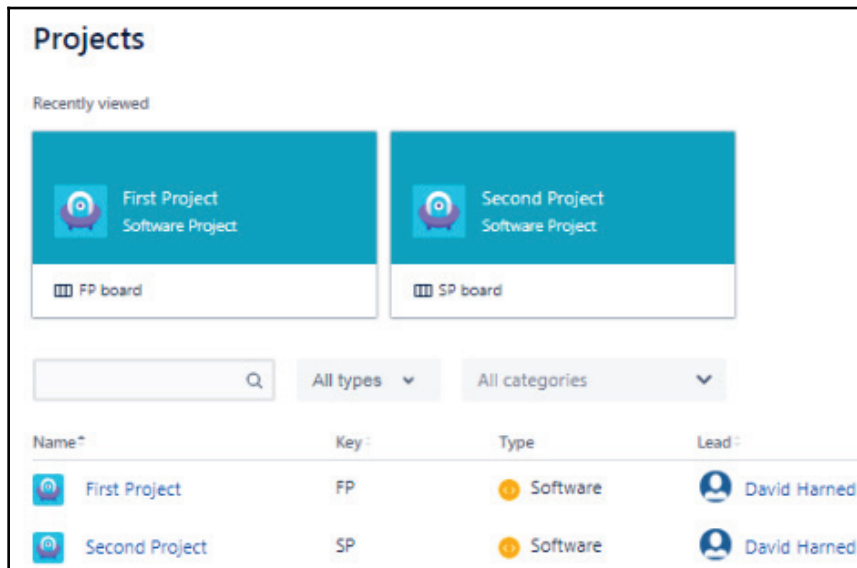
The fields themselves are the different types of information that are contained within the systems; these are the different attributes for the work types and more. The values are actually what's contained within those fields, so those are the actual values that we would be looking for. The operators are essentially the heart of the query—they're the intelligence—so they'd be things like equal to, or not equals to, or less than, and more, which we would then use to create some intelligence around the fields and values. And then, we've got keywords, which are really reserved words that we use in our query language to connect these different operators together.

Here is a link to a post on the Atlassian website that talks more deeply about searching JIRA: <https://confluence.atlassian.com/jiracore/blog/2015/07/search-jira-like-a-boss-with-jql>.







Simple and advanced JQL editors in JIRA

Let's take a look at the simple and advanced JQL editors in JIRA.

We have two projects called **First Project** and **Second Project** in our instance of JIRA. What we want to do is run some queries against these projects. Go to the **Issues** link in the upper left-hand corner:

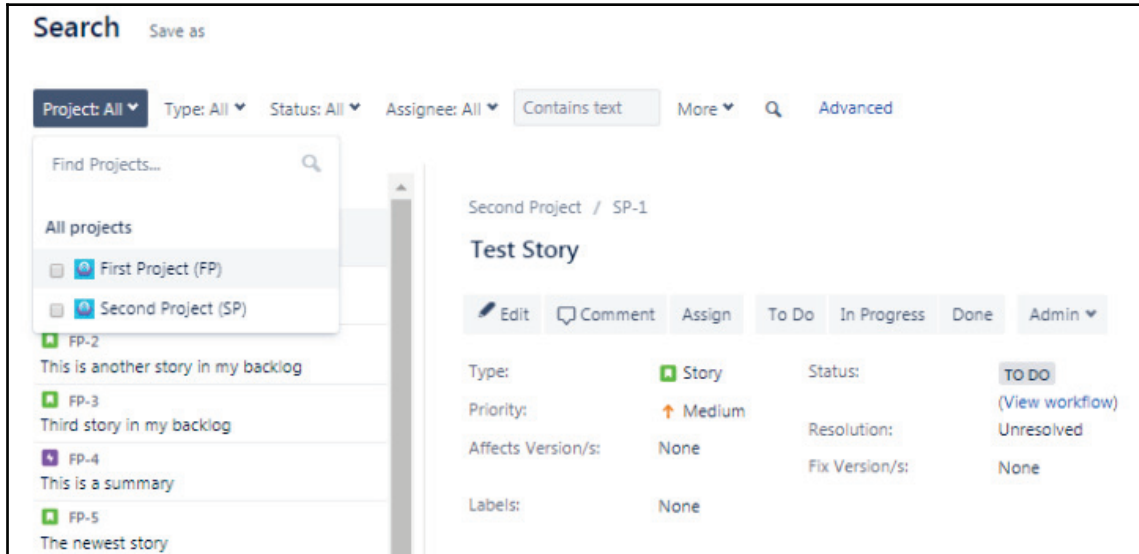


The screenshot shows the JIRA Projects dashboard. At the top, there's a search bar and filters for 'All types' and 'All categories'. Below that, there are two project cards: 'First Project' (Software Project) with an 'FP board' and 'Second Project' (Software Project) with an 'SP board'. At the bottom, there's a table listing the projects.

Name*	Key	Type	Lead
 First Project	FP	 Software	 David Harned
 Second Project	SP	 Software	 David Harned

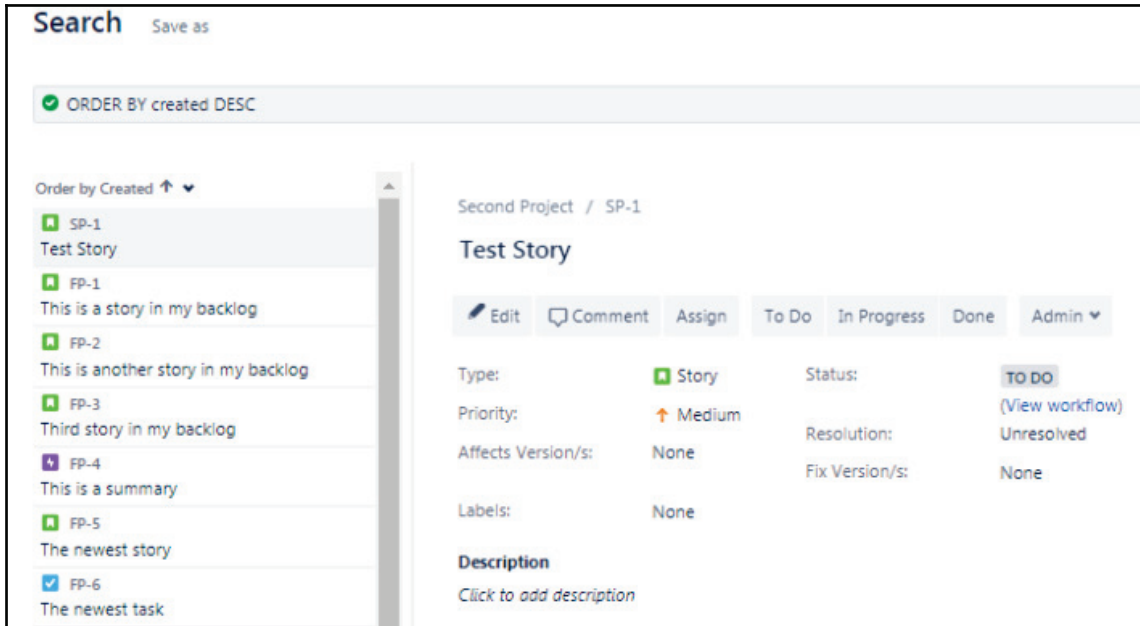
Dashboard containing WER projects

Our default screen for searching is going to be basically the most generic query that we can do, and we're actually in the advanced editor. Let's flip over to the basic editor in the upper right-hand corner of the screen:

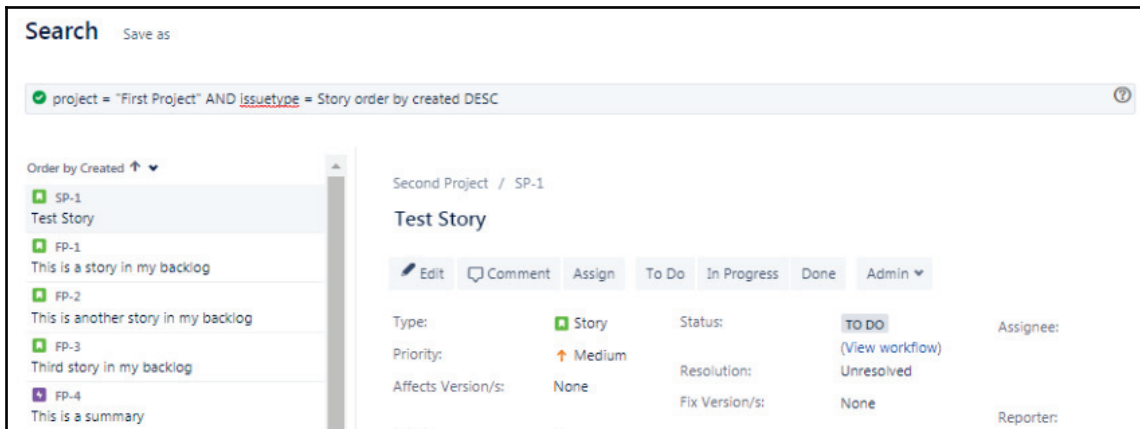


As we can see in the preceding screenshot, it's going to give us dropdowns that allow us to build a query. We know that we can check these projects. Say that we want to see just the **First Project**, or just the **Second Project**, and, as we do that, we can see the results underneath changing. We can also choose the types of issues that we want to see, the status that we want to see, who the issue is assigned to, and more. This is the basic editor, and we can run a query just by clicking and searching for issues.

If we go to the advanced editor, we can see that we can also look up some specific things:

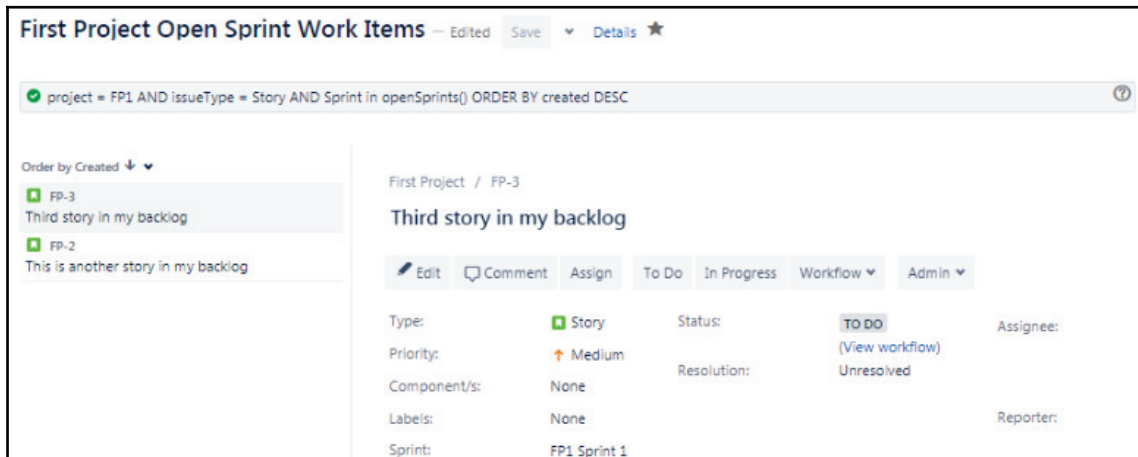


One of my favorite queries concerns open Sprints:



Here, we can say, *show me all of the open Sprints within the FP-1 project*. We can see that it's actually going to write a query for us, which makes it easier for us to run this. We can say that the *first project and our issue type equals story*, and that we want to search for it. We can see how this works; we can use the advanced editor to write our queries.

We also have a query that was already written, as shown in the following screenshot. This is the query for the **First Project** and the issue type is **Story**, which says *Show us everything that's in an open Sprint and order that by the created date descending*:



The screenshot shows a Jira issue view for 'Third story in my backlog' within the 'First Project / FP-3'. The query bar at the top displays: `project = FP1 AND issueType = Story AND Sprint in openSprints() ORDER BY created DESC`. The left sidebar shows a list of filters: 'FP-3 Third story in my backlog' and 'FP-2 This is another story in my backlog'. The main content area shows the issue details: Type: Story, Priority: Medium, Component/s: None, Labels: None, Sprint: FP1 Sprint 1, Status: TO DO (View workflow), Resolution: Unresolved, Assignee: (empty), and Reporter: (empty). Action buttons include Edit, Comment, Assign, To Do, In Progress, Workflow, and Admin.

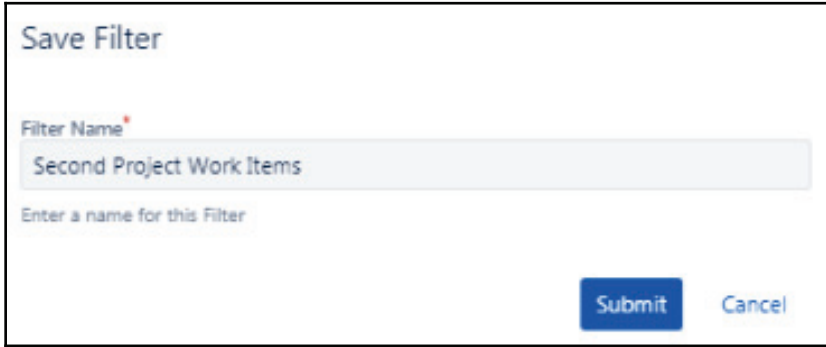
What we get back are two stories. These are the two stories that are currently in the **First Project** and that are in an open Sprint.

In this section, we looked at how we would write a query in both the basic and advanced query editor, and we've also been given a nice little sheet for the open Sprints.

Saving and managing filters

In this section, we're going to talk about saving and managing filters. This means that we're going to learn how to save a created filter, how to set writes for our saved filters, and how to see all of our saved filters and manage those.


Let's go ahead and hop over to JIRA and take a look at our issue searching screen. We have a query, so let's flip over to the basic editor and say *show me everything that's in the second project*. Saving this is really easy, as all we have to do is click on **Save as** and this will take you to the following screen, and write `Second Project Work Items` as our **Filter Name**:



The image shows a 'Save Filter' dialog box. At the top, it says 'Save Filter'. Below that is a text input field labeled 'Filter Name' with a red asterisk indicating a required field. The text 'Second Project Work Items' is entered into the field. Below the input field is a placeholder text 'Enter a name for this Filter'. At the bottom right, there are two buttons: 'Submit' and 'Cancel'.

Save Filter dialog

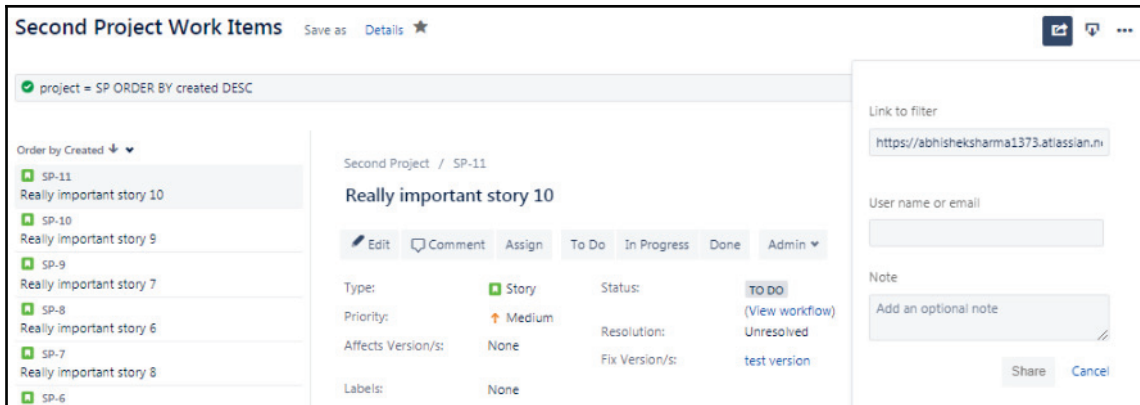
If we click **Submit**, we now have a saved filter. What we've done is we've taken a query and turned it into a filter by saving it so that we can run it again and again. If we look under **Manage filters** on the left, we can see the filter that we just saved to the **Second Project Work Items**:



The image shows a 'Filters' list in JIRA. At the top, there is a search bar and two dropdown menus for 'Owner' and 'Shared with'. Below that is a table with columns: Name, Owner, Shared with, and Popularity. There are three filters listed. The third filter, 'Second Project Work Items', is highlighted. A context menu is open over this filter, showing options: 'Manage subscriptions', 'Copy filter', 'Edit filter details', and 'Delete filter'.

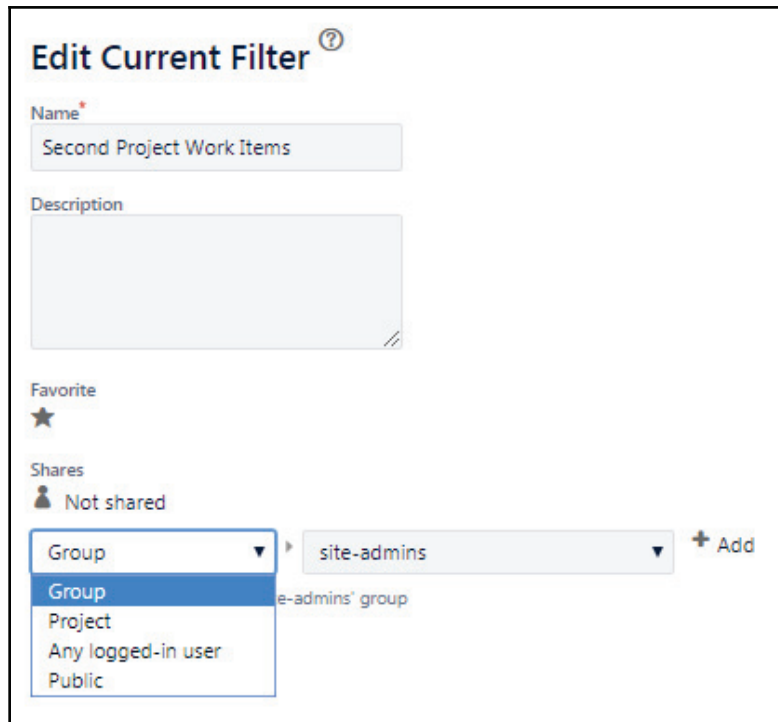
Name	Owner	Shared with	Popularity
Filter for FP board	David Hamed	project: First Project	0
Filter for SP board	David Hamed	project: Second Project	0
★ Second Project Work Items	David Hamed	Private filter	1

We can also see that we have three ellipses on the right-hand side, which provide us with some options such as **Manage subscriptions**, **Copy filter** and **Edit filter details**, and we can even click on **Shared with** and take a look at who these are shared with. If we click on the **Second Project** work items, we're going to see the content of that query. We can also click **Advanced** so that we can see what query language we used:



Second Project Work Items

In the top corner of the preceding screenshot, we can see that we can share the item by providing the link, inserting the username or email of that person contained within JIRA, and then we can add a note and hit **Share**. We've also got the details of the item that we created:



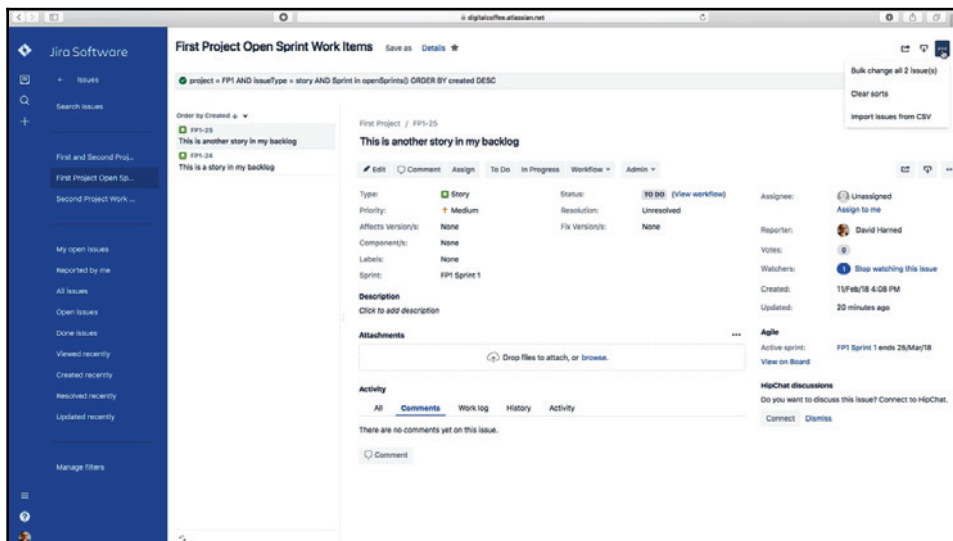
This allows us to set the permissions and subscriptions, and see who should have access to them and be able to see the content of this filter. We can see that we have the ability to share this with a group, a particular project, any logged-in user, or just make it public. That's how we save and we manage our queries and turn them into filters.

Executing bulk changes

In this section, we're going to be talking about executing bulk changes. This can be really powerful when we need to make a lot of changes to many items and we don't want to make those changes individually. We do this by figuring out whether there is a pattern or a way that we can make that change to many items and execute what's called a bulk change. We're going to learn how to execute a bulk change using the results of a filter.

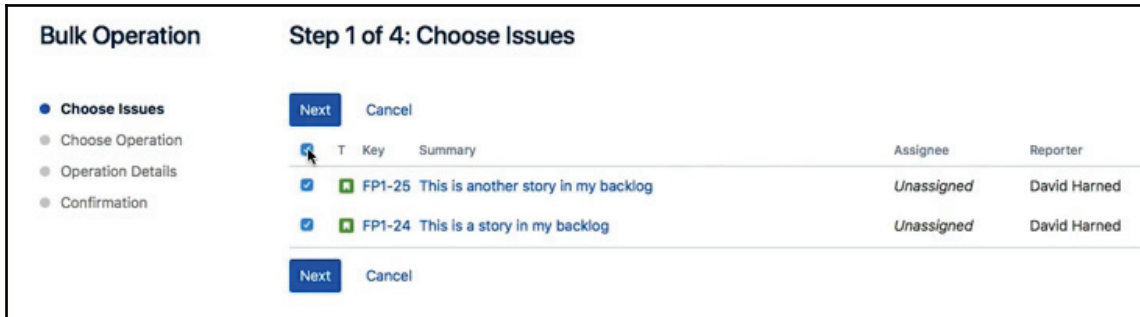
Let's jump back over to JIRA and take a look at one of our filters, as shown in the following screenshot. In our **First Project Open Sprint Work Items** filter, we can see that this says project FP1, and, if the issue type is story, we're going to look at our open Sprints and return everything ordered by the created date descending. It returns the two stories, **FP1-24** and **FP1-25**.

We also want to change both of these stories to tasks. The best way to do this would be with a bulk change, so let's proceed. We're going to go to the ellipsis in the right-hand corner, and we're going to click on **Bulk change all 2 issue(s)**:



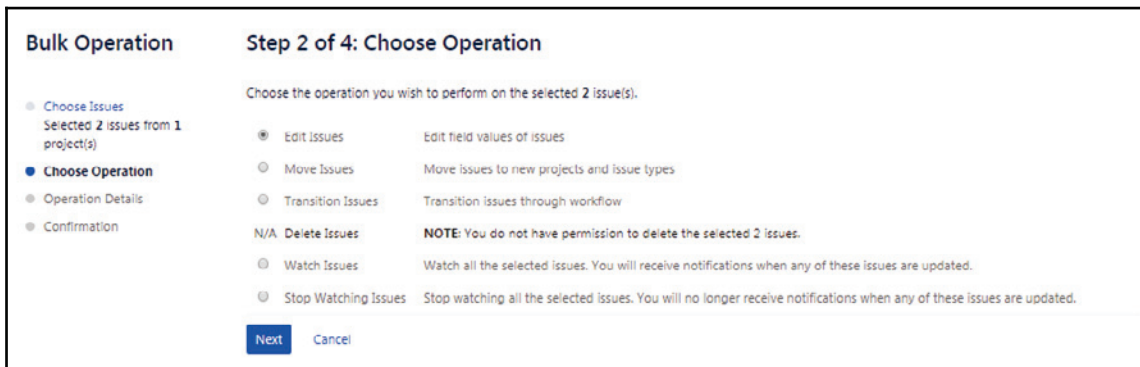
Executing bulk changes

This will bring us to the following screen:



Bulk operation

We're going to select each of the issues that we want to change, and, in this case, we can just select the top one. It will check all of them, so we can click **Next**. This will take us to the following screen:



We're going to choose the operation we want to perform. We want to edit these issues, but we can also move them from one project to another, transition them to the workflow, and more. Let's click **Edit Issues**, and then **Next**, and then we're going to change the issue type to **Task**:

- Choose Issues
Selected 2 issues from 1 project(s)
- Choose Operation
- **Operation Details**
- Confirmation

Step 3 of 4: Operation Details

Choose the bulk action(s) you wish to perform on the selected 2 issue(s).

- Change Issue Type
- Change Priority
- Change Component/s
- Change Assignee
- Change Environment
- Change Due date
- Change Comment
- Change Labels
- Change Approvers
- Change Change completion date

Task

?

Medium

?

Add to existing

▼

Start typing to get a list of possible matches or press down to select.

Automatic

▼

Assign to me

Style▼

B I U A ^A

For example operating system, software platform and/or hardware specifications (include as appropriate for the issue).

Style▼

B I U A ^A

Add to existing

▼

Begin typing to find and create labels or press down to select a suggested label.

Start typing to get a list of possible matches.
Contains users needed for approval. This custom field was created by Jira Service Desk.

Specify the completion time for the change request

As we can see in the preceding screenshot, we can also change lots of other items as well. We'll click **Next**, and then we can click **Confirm**, as shown in the following screenshot:

Bulk Operation Step 4 of 4: Confirmation

- Choose Issues
Selected 2 issues from 1 project(s)
- Choose Operation
- Operation Details
- **Confirmation**

Updated Fields

Field Name	Field Action	Field Value
Issue Type	Change to	<input checked="" type="checkbox"/> Task

⚠ Email notifications will be sent for this update.

ℹ The above table summarizes the changes you are about to make to the following 2 issues. Do you wish to continue?

T	Key	Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated	Due
<input checked="" type="checkbox"/>	FP-2	This is another story in my backlog	Unassigned	David Harned	↑	TO DO	Unresolved	12/Jul/18	16/Jul/18	
<input checked="" type="checkbox"/>	FP-3	Third story in my backlog	Unassigned	David Harned	↑	TO DO	Unresolved	12/Jul/18	16/Jul/18	

And it will go ahead and execute that change for us as follows:

Bulk Operation

Bulk Operation Progress

Editing 2 issues

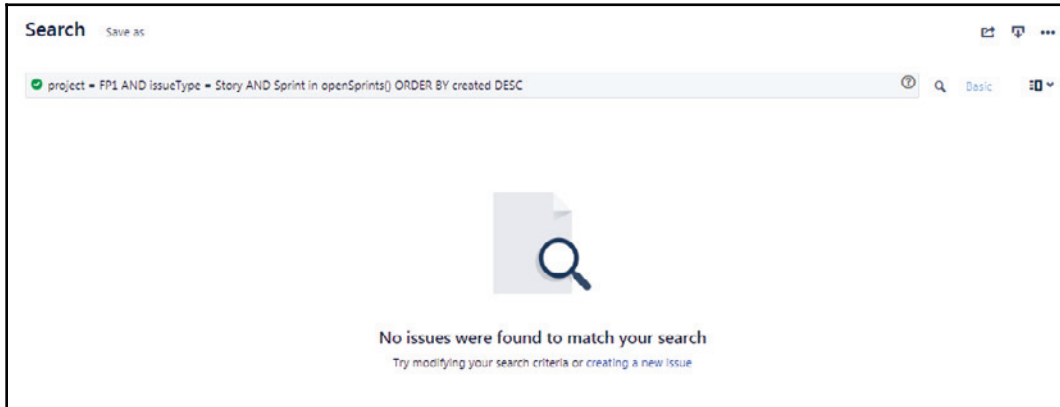
Bulk operation is 100% complete.

Task completed in 0 seconds

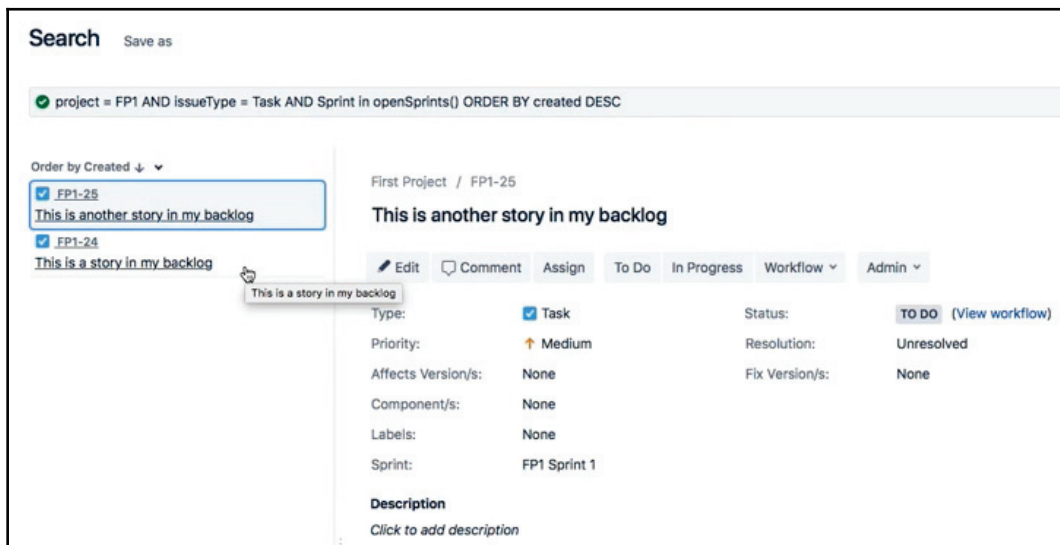
Started Today 8:19 PM.

Finished Today 8:19 PM.

While it's running, we'll let it run for a minute and, when it's done, we can click **Acknowledge**. If we go back to our query that was for this project issue type story, we get no results:



If we change the story to a Task and run this query again, we can see that we've now changed these items from stories into tasks:



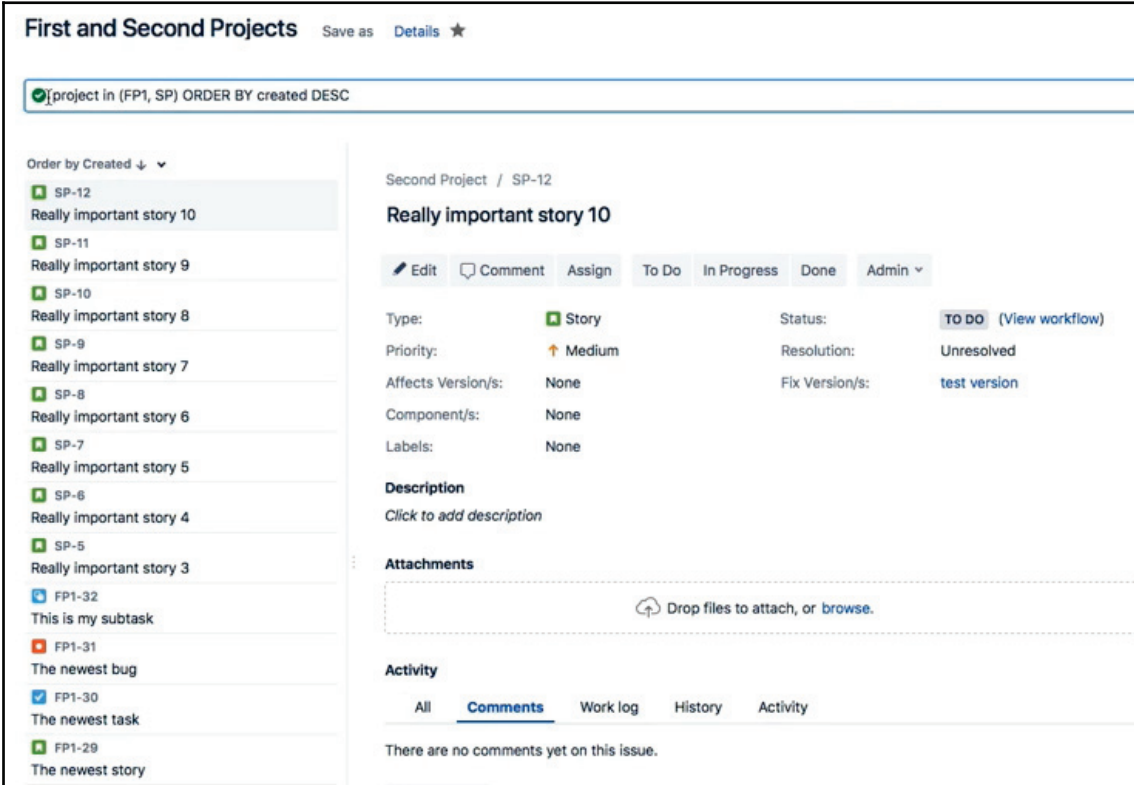
Running query

That's how we execute a bulk change.

Creating new boards from saved filters

In this section, we're going to be talking about trading new boards from saved filters. Here, we're going to use a saved filter and create a new JIRA board. In the previous sections, we've talked about what JQL is, creating a filter from a query, and bulk changes, and now we're going to take a look at using those same filters to create a JIRA board. There's actually a lot of power within JIRA.

As we can see in the following screenshot, we have a query. We've actually saved this query as a filter. Let's take a look at what this query does:



The screenshot shows a JIRA issue view for 'Really important story 10' in the 'Second Project' (SP-12). The issue is a 'Story' with a 'Medium' priority, 'Unresolved' status, and 'test version' fix version. The left sidebar shows a list of issues ordered by creation date descending, including items from 'Second Project' (SP-12 to SP-5) and 'First Project' (FP1-32 to FP1-29). The main content area shows the issue details, description, attachments, and activity sections.

First and Second Projects Save as Details ★

👍 [project in (FP1, SP) ORDER BY created DESC

Order by Created ↓

- SP-12 Really important story 10
- SP-11 Really important story 9
- SP-10 Really important story 8
- SP-9 Really important story 7
- SP-8 Really important story 6
- SP-7 Really important story 5
- SP-6 Really important story 4
- SP-5 Really important story 3
- FP1-32 This is my subtask
- FP1-31 The newest bug
- FP1-30 The newest task
- FP1-29 The newest story

Second Project / SP-12

Really important story 10

Edit Comment Assign To Do In Progress Done Admin ▾

Type: Story Status: TO DO (View workflow)

Priority: Medium Resolution: Unresolved

Affects Version/s: None Fix Version/s: test version

Component/s: None

Labels: None

Description

Click to add description

Attachments

Drop files to attach, or browse.

Activity

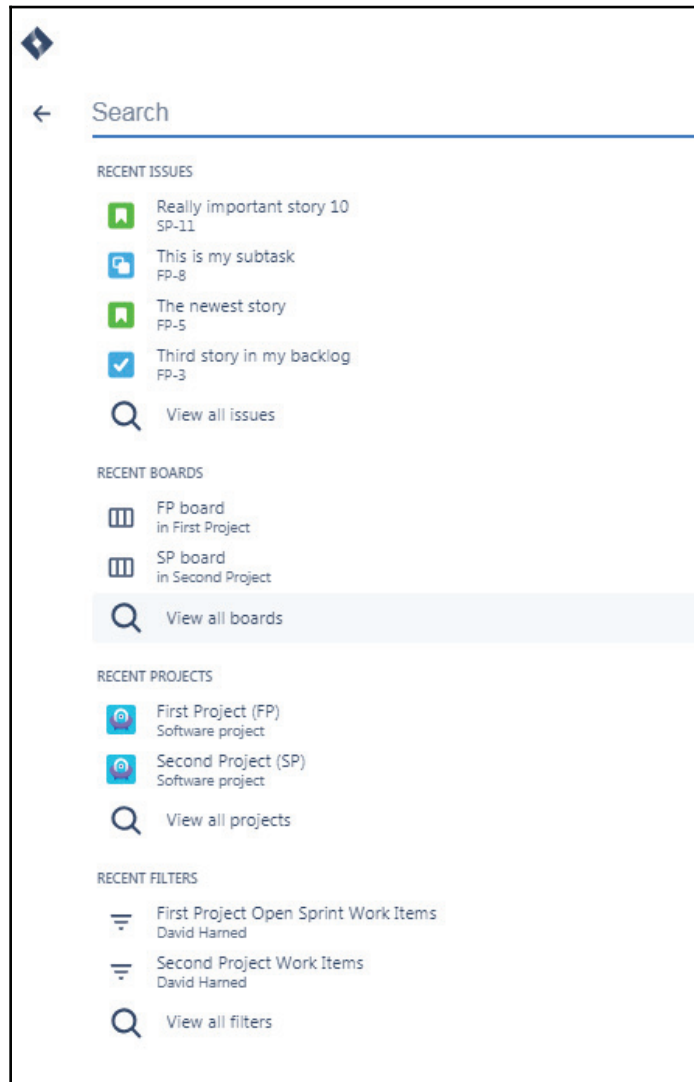
All Comments Work log History Activity

There are no comments yet on this issue.

Saved filters

As we can see in the preceding screenshot, it's basically returning all of the items in the **First Project** and **Second Project**, and it's ordering them by the creation date descending. That's pretty simple; it's basically saying, *show me all of the work items in these two projects.*

Let's use this to create a new board. Let's say, hypothetically, that we want to go and mine these two projects together into a bigger initiative, and we want to create a board that will let us view all of that in a single board view. What we'll do now is go over to the search screen, to **RECENT BOARDS**. We can see in the following screenshot that we have **View all boards**:



We might remember that a board gets created automatically when we create a new project. Here, we can see that we actually have two boards already; we have the first project and second project boards as follows:

Name	Type	Admin	Location
FP board	Scrum	David Hamed	First Project (FP)
SP board	Scrum	David Hamed	Second Project (SP)

We're now going to create a new project, so if we go to the upper right-hand corner, we can see that we have a **Create board** option, so let's click that. We're going to get a prompt that gives us some options on what kind of board we'd like to create:

Create a board

Agility

Agility provides an easy, simplified board that delivers more power as you need it, enabling your team to get up-and-running fast.

Create an agility board

Scrum

Scrum focuses on planning, committing and delivering time-boxed chunks of work called Sprints

Create a Scrum board

[Create a Scrum board with sample data](#)

Kanban

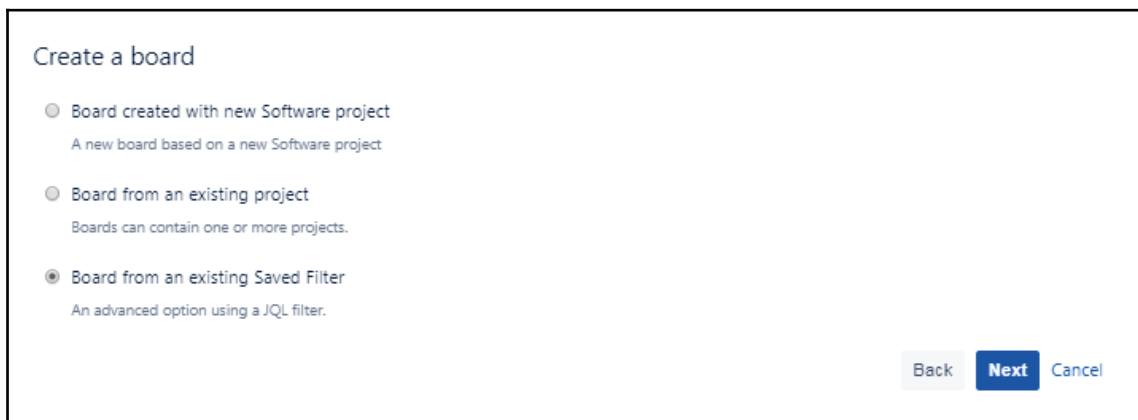
Kanban focuses on visualising your workflow and limiting work-in-progress to facilitate incremental improvements to your existing process

Create a Kanban board

[Create a Kanban board with sample data](#)

[Cancel](#)

The first is the **Agility** board, which is the board that JIRA will create to allow us to view the content of a project in a more streamlined way. We already have projects in place with boards, and so we're trying to create a board from a query. We can also see that we have a **Scrum** board, which would include things like iterations and similar. We're really just trying to bring a board view together that gives us all of the contents of the two projects. **Kanban** is going to be what we want to use in this situation. Kanban is really about moving work items and a limited number of work items through a workflow, with each column representing a workflow. Let's create a Kanban board. We can see in the following screenshot that we get a few more options, for example, do we want to create a new software project, an existing project, or a board from an existing saved filter?



Let's create a board by clicking on the **Board from an existing Saved Filter** option and hitting **Next**. We'll call this board `All Projects`, and we're going to use the **Saved Filter** of the **First and Second Projects**. We're not going to share it now, but we're going to have me as the owner, and then, as a location, we can save this as another board underneath either the first or second projects, or we can just attach this to our profile, as shown in the following screenshot:

Name this board

Board name*

Saved filter*

Location*

Select a Software project or your own profile as the place where this board will live

Saved Filters

Choose from a list of existing filters as a base for your new board. To create a new Saved Filter, save a search in the Issue Navigator.

Back Create board Cancel

Let's go ahead and create the board. We can see that we have a Kanban board with three columns, **TO DO**, **IN PROGRESS**, and **DONE**, and all of the content of both of the projects:

All Projects

Kanban board

Release ▾ ...

Quick filters ▾ Assignee ▾

TO DO 5	IN PROGRESS 0	DONE 3
<p>FP-5 The newest story</p> <p>This is my subtask</p> <p>FP-8</p> <hr/> <p>The newest bug</p> <p>My Test Epic</p> <p>FP-7</p> <hr/> <p>The newest task</p> <p>My Test Epic</p> <p>FP-6</p> <hr/> <p>The newest story</p> <p>My Test Epic</p> <p>FP-5</p> <hr/> <p>This is a summary</p> <p>My Test Epic</p> <p>FP-4</p>	<p>IN PROGRESS 0</p>	<p>Third story in my backlog</p> <p>FP-3</p> <hr/> <p>This is another story in my backlog</p> <p>FP-2</p> <hr/> <p>This is a story in my backlog</p> <p>FP-1</p>

We can see in the preceding screenshot, which shows us that all of the first and second project's content is now contained within this board, and that we live in this workflow. If you found that easy, we will now look at a few options that give us both of those projects combined.

Summary

In this chapter, we learned what the JIRA query language is, that is, JQL— this is not the Java Query Language, but the JIRA Query Language. We learned how to write both simple and advanced queries using JQL, how to export our query results, and we learned how to save the query and turn it into a filter. Finally, we learned how to use those saved filters to execute bulk changes, and we learned how to use a saved filter to create a new JIRA board.

In our next and final chapter, we're going to be talking about dashboards and widgets.

6 Dashboards and Widgets

In this chapter, we're going to talk about dashboards and widgets in JIRA. We're going to cover creating and managing dashboards, adding gadgets to our dashboards, and sharing our dashboards.

Dashboards are really about broadcasting how things are going and being able to share that at a high level, and being visible and transparent with the results of the project.

In this chapter, we will cover the following topics:

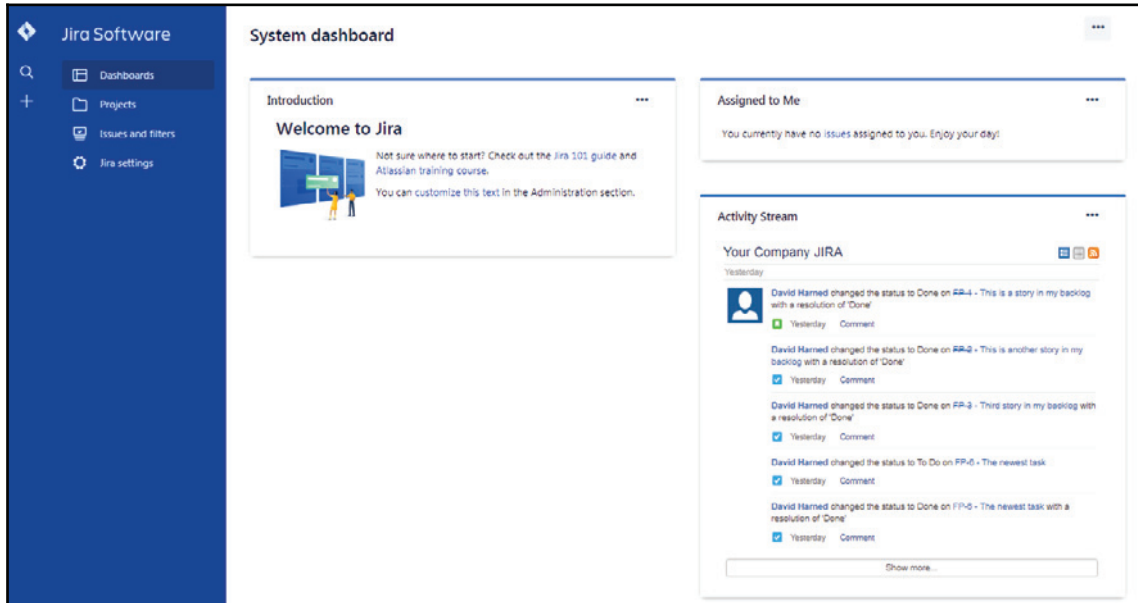
- Creating and managing a dashboard
- Adding gadgets to our dashboard
- Sharing our dashboard

Creating and managing a dashboard

In this section, we're going to talk about creating and managing a dashboard. We're going to learn what a dashboard is and how to create a dashboard.

Let's go ahead and hop on over to JIRA, and take a look at some dashboards.

In our projects, we had **First Project** and **Second Project**, and on the left-hand side, we can see **Dashboards**. JIRA will come with a **System dashboard** by default, so we can take a look at that in the following screenshot:



System dashboard

What we can see is an **Activity Stream** and other tabs, but what we want to do is create our own dashboard, so in the upper right-hand corner go to **Create dashboard**. We'll get some options, where we can name the dashboard *My Great Dashboard*, and give it a description of *All the good things*. We can **Start from a Blank dashboard, System dashboard**, or an existing one. We have a favorites list, and we can mark our dashboard as a favorite in case we have many. We can also choose our sharing options, where we've got project, any login user, or public. However, for now, we'll just leave it in its default state:

Create dashboard

Name*
My Great Dashboard

Description
All the good things.

Start from
Blank dashboard
Blank dashboard
System dashboard

Favorite
★

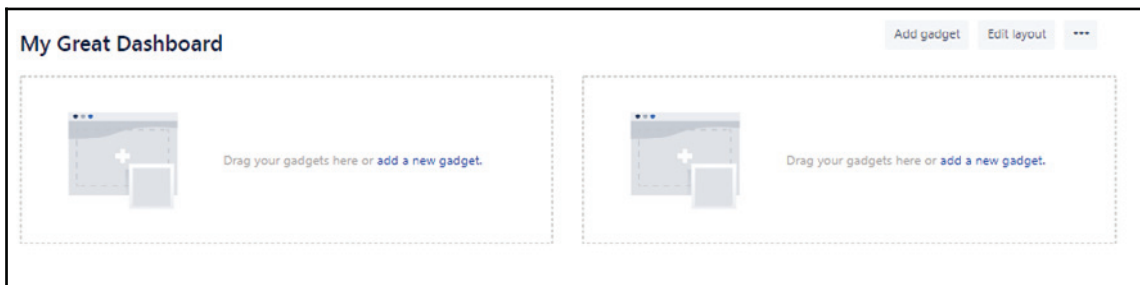
Shared with
Not shared

Group site-admins + Add

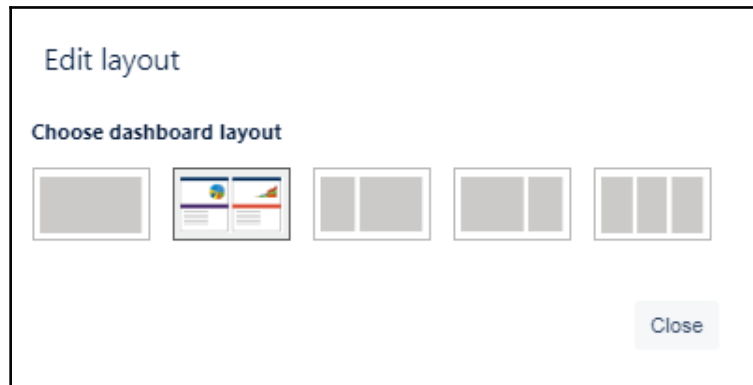
Shared with everyone in the 'site-admins' group

Create Cancel

As we can see in the following screenshot, we've created our dashboard, where we can add a new gadget:



But more importantly at this moment in time, let's take a look at the layout:



We can see in the preceding screenshot that we can have one page, which is like one giant chunk where we can have multiple columns: we can have a small column and a big one, we can have the opposite of that, or we can have three columns; it's up to us what we choose. We'll leave it as the default settings right now, but we think it's important that we know that we have these options so that we can share a dashboard that is configured in a way that's visually pleasing once we start to put these gadgets together. That's how we create a dashboard.

Adding gadgets to our dashboard

In this section, we're going to talk about adding gadgets to our dashboard. We'll learn about how to add the filter results gadget to our dashboard, the pie chart gadget, and also the two-dimensional filter statistics. In reality, there's lots and lots of gadgets.

Let's go ahead and add some gadgets. Go back to JIRA; remember, we were dealing with my great dashboard, which we created in the last section.

There's something we should know about gadgets. First of all, there are lots of them, so JIRA comes pre-loaded with quite a few. Also, when we took a look at the Atlassian marketplace, there were about 95 different gadgets that were available for us to load into our dashboard as well. There's even a gadget there to help us when we create our own gadgets, so in addition to that, we can do things like reporting. This means that we can really extend a dashboard quite a bit to visualize all kinds of different things.


Let's take a look at the gadgets we want to add. On our dashboard, we've got the two column layout, so let's click **add a new gadget**. This will take us to the following screen:

Add a gadget

Search

CATEGORIES

- All 33
- Charts 12
- JIRA 32
- Wallboard 7




Activity Stream

By Atlassian • Local

Lists recent activity in a single project, or in all projects.

[Show XML link](#)

Add gadget




Agile Wallboard Gadget

By Atlassian • Local

Displays a board as a Wallboard gadget

[Show XML link](#)

Add gadget




Assigned to Me

By Atlassian • Local

Displays all unresolved issues assigned to me

[Show XML link](#)

Add gadget




Average Age Chart

By Atlassian • Local

Displays the average number of days issues have been unresolved.

[Show XML link](#)

Add gadget




Average Number of Times in Status

By Atlassian • Local

Displays the average number of times issues have been in a status.

[Show XML link](#)

Add gadget



Average Time in Status

By Atlassian • Local

Displays the average number of days issues have spent in status.

Add gadget

[Close](#)

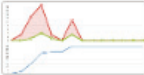
Adding gadgets

The first one we're going to talk about today is used to filter results. We can see there's lots of different things available on the left-hand side; we've got **Wallboard**, **JIRA**, **Charts**, gadgets, and more. We're going to be looking for the filter results, so we'll search by *filter*. Now, add the gadget:

Add a gadget

CATEGORIES

- All** 10
- Charts 8
- JIRA 10
- Wallboard 0



Created vs. Resolved Chart
By Atlassian • Local Add gadget

Displays created issues vs. resolved issues for a project or saved filter.

[Show XML link](#)


JQL Issues	372
JQL Unresolved Issues	6
Test Current User	BL

[Create Filter](#) [Manage Filters](#)

Favorite Filters
By Atlassian • Local Add gadget

Lists favorite filters for the current user.


[Show XML link](#)



Filter Results
By Atlassian • Local Add gadget

Shows the issues/results for a saved filter.


[Show XML link](#)



Heat Map
By Atlassian • Local Add gadget

Displays the matching issues for a project or filter as a heat map.


[Show XML link](#)



Pie Chart
By Atlassian • Local Add gadget

Displays the matching issues for a project or filter as a pie chart.

[Show XML link](#)



Resolution Time
By Atlassian • Local Add gadget

Displays a bar graph of elapsed time to resolve issues for a

[Close](#)

Adding gadgets

[129]


EBSCOhost - printed on 2/9/2023 11:41 AM via . All use subject to <https://www.ebsco.com/terms-of-use>

We're also going to look for the pie chart, so we type in `pie`, and add that gadget from the result as follows:

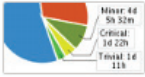
Add a gadget

CATEGORIES

- All 2
- Charts 2
- JIRA 2
- Wallboard 0



Pie Chart
By Atlassian • Local
Displays the matching issues for a project or filter as a pie chart.
[Show XML link](#)



Workload Pie Chart
By Atlassian • Local
Displays the matching issues for a project or filter as a pie chart.
[Show XML link](#)

[Close](#)

And then we're also going to get the two-dimensional filter statistics gadget, so let's take a look at that one:



Adding the two-dimensional filter statistics gadget

Those are the three we're going to look at, and as we can see, we can keep adding them to our dashboard. The first thing we're going to do is give it a filter, which is going to be the two-dimensional filter, and we want to say, *we want to see the contents of the first and second projects*. Remember that filter from the previous section? This will show us the contents of all of the first and second projects. On our **XAxis**, we want to have the **Status** as horizontal access, so this would be the status of all the work items, and on the **YAxis**, we want to have the **Assignee**. We can change the sort, the sort direction, the number of results, and more. We can even set it to update every 15 minutes if we want to. We'll leave these as default for now and click **Save**:

Two Dimensional Filter Statistics ⋮

Saved Filter*
First and Second Project ▼

Advanced Search

XAxis*
Status ▼

YAxis*
Assignee ▼

Sort By*
Natural ▼

Sort by row total or natural field order.

Sort Direction*
Ascending ▼

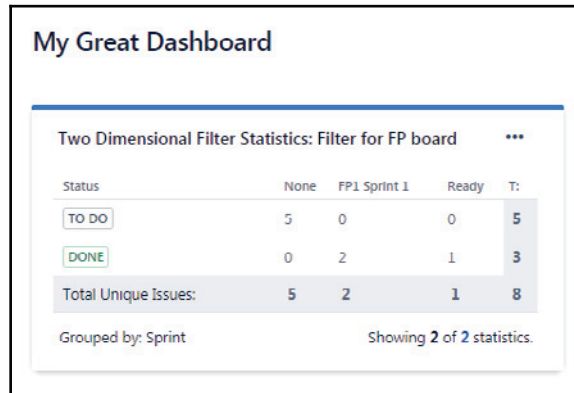
Number of Results*
5

Number of results to display.

Auto refresh
 Update every 15 minutes

Save Cancel

What we're going to see is the contents of the first and second projects. Here, we can see the **Assignee**, or unassigned items, how many items are in progress, how many are to do, how many are done, and we get our totals. The other interesting thing is that we can actually click the link and this will take us to search for those issues, and it will show us the 18 items that are our items that are done. If we have a large team, this is a really good way at looking at the items for each person and the state that they're in. This is why we can use this one quite a bit:



Let's take a look at the pie chart. We can do the same thing here: we can take a look at some of the filters and the projects that we have:

Pie Chart

Project or Saved Filter:
First Project
 Search

Project or saved filter to use as the basis for the graph.
 Advanced Search

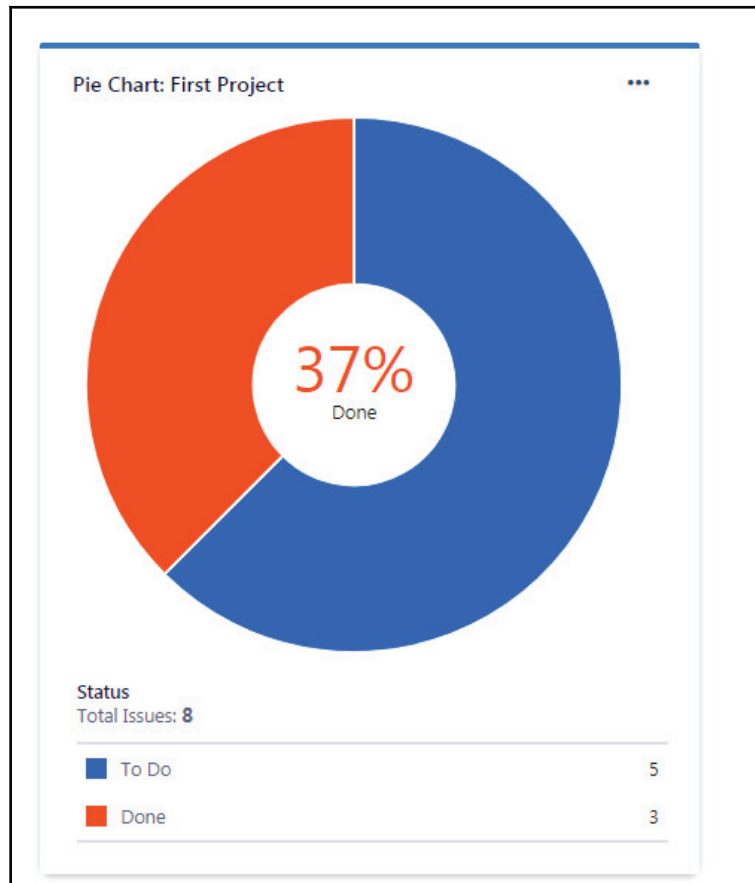
Statistic Type:
 Status

Select which type of statistic to display for this filter.

Auto refresh
 Update every 15 minutes

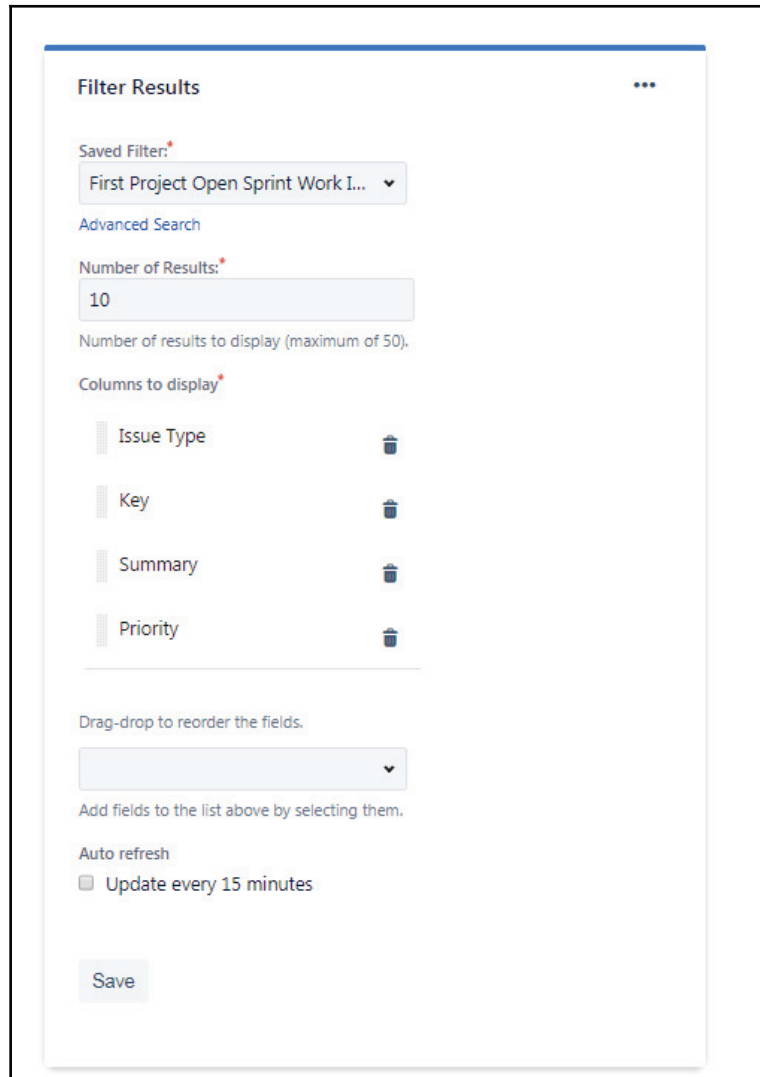
Save Cancel

Let's take a look at what we have. We will look at the **First Project** and keep the **Statistic Type** as **Status**. We'll hit **Save**, and we can see that we now have a pie chart:



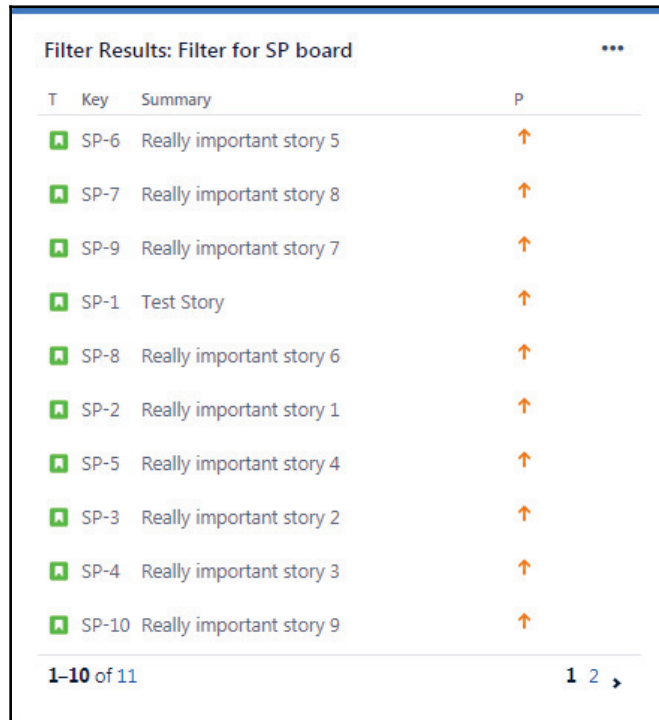
This chart shows us the work that's done, and the work that's still to be done. If we had a large team, for example, we could look at how many people have, how many work items, or what the status is of the items that they have, or any number of things, but we will be able to visualize this in a pie chart.

Next, we're going to take a look at our favorite filter. Let's pick a filter. We can pick the open Sprint work items option that we have. Again, we're going to use the First Project items that are open. Here, we can have up to 10 results; there are the columns to display, and we can configure them or delete them. Let's go ahead and save it:



The screenshot shows a 'Filter Results' configuration window. At the top, it says 'Filter Results' with a three-dot menu icon. Below that, there's a 'Saved Filter:' dropdown menu currently set to 'First Project Open Sprint Work I...'. Underneath is a link for 'Advanced Search'. The 'Number of Results:' is set to '10' in a text input field, with a note below it stating 'Number of results to display (maximum of 50)'. The 'Columns to display:' section lists four items: 'Issue Type', 'Key', 'Summary', and 'Priority', each with a trash icon to its right. Below this list is a 'Drag-drop to reorder the fields.' section with an empty dropdown menu. A note says 'Add fields to the list above by selecting them.' There is an 'Auto refresh' section with a checkbox for 'Update every 15 minutes' which is currently unchecked. At the bottom left, there is a 'Save' button.

Let's go ahead and go back into our edit. Here, we can take a look at the **Second Project** board. We can select a filter, and then we can display the contents of that filter, shown as follows:



The screenshot shows a Jira board filter results window titled "Filter Results: Filter for SP board". It displays a list of 10 items, each with a green icon, a key, a summary, and a priority indicator (orange arrow pointing up). The items are sorted by priority. At the bottom, it shows "1-10 of 11" and a pagination control with "1 2" and a right arrow.

T	Key	Summary	P
	SP-6	Really important story 5	↑
	SP-7	Really important story 8	↑
	SP-9	Really important story 7	↑
	SP-1	Test Story	↑
	SP-8	Really important story 6	↑
	SP-2	Really important story 1	↑
	SP-5	Really important story 4	↑
	SP-3	Really important story 2	↑
	SP-4	Really important story 3	↑
	SP-10	Really important story 9	↑

We can see the contents because we selected 10 items and we're only seeing 10 of 12, but we could set that to as many as we wanted as well as configure which columns show up.

We could use these three examples of gadgets on our dashboards, but there are lots of different ones that we can choose from. In the next section, we're going to be talking about sharing our dashboard.

Sharing the dashboard

In this section, we're going to learn how to share our dashboard. Remember, the dashboard itself and the whole concept behind it is about broadcasting results and making the results of the effort visible and transparent, so that's really what we're trying to do: make sure that everyone has access to them.

Let's hop on over to JIRA and take a look at the sharing settings for the dashboard. In the previous section, we created a migrated dashboard, and we gave it the contents that we can see in the following screenshot. We want to share this out so that everyone can see it. Go to the ellipsis in the upper right-hand corner, click it, and say **Share dashboard**:

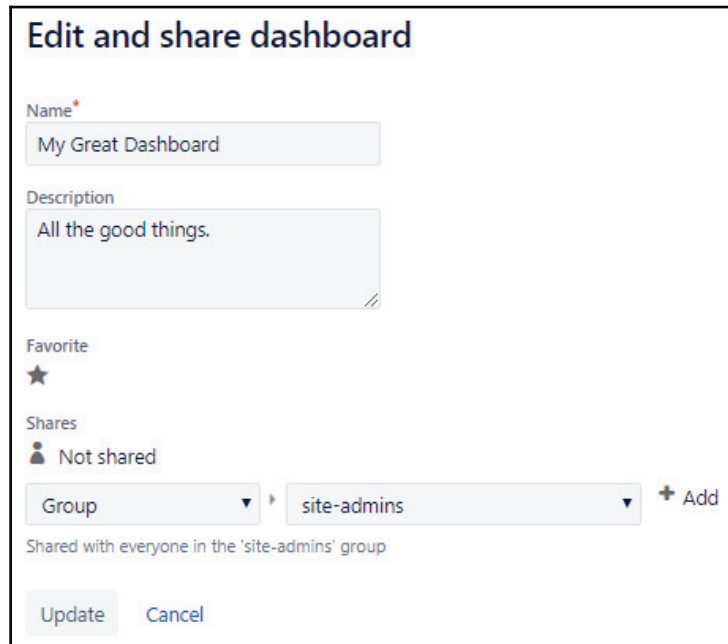
The screenshot shows a JIRA dashboard titled "My Great Dashboard". It contains two main widgets:

- Two Dimensional Filter Statistics: Filter for FP board**: A table showing issue counts for different statuses across sprints.
- Pie Chart: First Project**: A donut chart showing the distribution of issues in a project.

A context menu is open in the top right corner, listing various actions for the dashboard. The "Share dashboard" option is highlighted.

Status	None	FP1 Sprint 1	Ready	T:
TO DO	5	0	0	5
DONE	0	2	1	3
Total Unique Issues:	5	2	1	8

We also have lots of other options as well, but here, we're going to look at share. We can see that we've got the settings that we had when we originally set it up:



The screenshot shows a dialog box titled "Edit and share dashboard". It contains the following fields and options:

- Name:** A text input field containing "My Great Dashboard".
- Description:** A text area containing "All the good things."
- Favorite:** A star icon.
- Shares:** A section with a person icon and the text "Not shared". Below this is a dropdown menu set to "Group", followed by a right-pointing arrow, another dropdown menu set to "site-admins", and a "+ Add" button.
- Summary:** Below the share settings, it says "Shared with everyone in the 'site-admins' group".
- Buttons:** At the bottom, there are "Update" and "Cancel" buttons.

We can choose who should be able to see the dashboard, and we can even make it **Public**. We can see that this page states, **sharing with the public will make this visible to everyone including users who are not logged in**, but if we want our customers or someone to be able to see this, this is a nice option, and we can also click **Update** and share with them that way. Otherwise, we could choose anyone on the project, and then select which project, and more. This is basically how we would share our dashboard.

Summary

In this chapter, we talked about how to create a dashboard, how to add gadgets to our dashboard using the filters that we created in the previous sections, and how to share our dashboard with others and make those results transparent and visible.

This brings us to the end of our JIRA time together.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



Jira Software Essentials - Second Edition

Patrick Li

ISBN: 9781788833516

- Understand the basics and agile methodologies of Jira software
- Use Jira Software in a Scrum environment
- Manage and run Jira Software projects beyond the out of box Scrum and Kanban way
- Combine Scrum and Kanban and use other project management options beyond just agile
- Customize Jira Software's various features and options as per your requirements
- Work with Jira Agile offline, and plan and forecast projects with an agile portfolio
- Integrate Jira Agile with Confluence and Bitbucket



JIRA 7 Administration Cookbook
Second Edition

Over 80 hands-on recipes to help you efficiently administer,
customize, and extend your JIRA 7 implementation

Patrick Li [PACKT] enterprise

JIRA 7 Administration Cookbook - Second Edition

Patrick Li

ISBN: 9781785888441

- Customize basic settings for your projects, such as screens and fields
- Create and customize workflows to suit your business process needs
- Make workflows more effective and efficient
- Manage users and groups inside JIRA and manage advanced login options
- Secure your JIRA instance using effective practices
- Perform e-mail functionalities with JIRA
- Extend JIRA to integrate with other products and services

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

A

Atlassian

account, creating 10, 12, 13

B

backlog

items, managing 48, 50, 51, 53, 56

board

configuring 56, 58, 60, 62, 64, 65

creating 57, 58, 60, 62, 64, 65

creating, from filters 118, 121

creating, with filters 123

bugs 31

bulk changes

executing 113, 114, 116, 117

burndown report

about 95, 97, 98

examples 95, 96

D

daily Scrum 75, 78, 81

dashboard

creating 124, 127

gadgets, adding 127, 129, 132, 134, 136

managing 124, 127

sharing 137, 138

E

epic burndown 101

epic reports 103

epics

about 31

creating 32, 36

F

filters

boards, creating 118, 121, 123

managing 110, 112, 113

saving 110, 112, 113

G

gadgets

adding, to dashboard 127, 129, 132, 134, 136

I

issue searching

JQL, using 106

issue type attributes

adding 40, 42, 43, 45

removing 48

J

JIRA Query Language (JQL)

advanced editors 107, 109, 110

used, for issue searching 106

JIRA

about 5, 82

projects, using 6, 9

P

projects

creating 14, 16

managing 14

notifications 28

permissions 27

setting up, with notifications 29

setting up, with screens 22, 24

setting up, with workflows 24, 26

workflows 18, 20, 21

R

release burndown
 about 101
 examples 101, 103
releases 89, 91, 93, 94

S

Sprint report 98, 99
Sprint
 closing 85, 86, 87
 creating 67, 70, 72, 75
 initiating 67, 70, 72, 75
stories
 about 31
 creating 37, 38, 40
Story Points 100

T

t-shaped people 82

tasks

 about 31
 creating 40

U

user stories 31

V

velocity chart
 about 100
 example 100, 101
version reports
 about 103
 example 104, 105
versions 89, 91, 93, 94

W

work
 organizing 81, 84