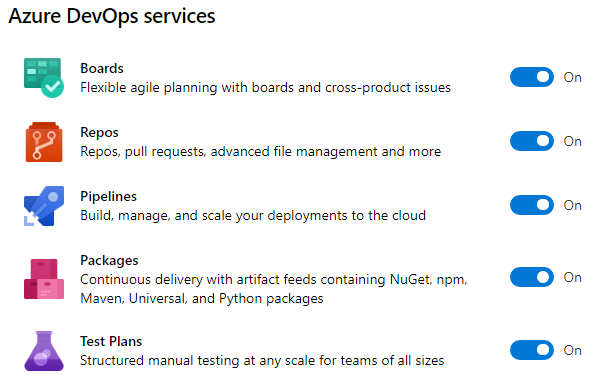
# Chapter 4: Tracking Work

Now that you’ve looked at the capabilities of the professional DevOps environment and a mix of tools that can be a part of it, we’ll drill down into each product within the Azure DevOps family and set it up in the proper way. You’ll certainly want to customize the configuration, but your suggested configuration works great in 80% of the cases. If you’ve already read the book “The Phoenix Project” by Kim, Spafford, and Behr, you’ll recognize the principles we implement in this chapter. You might want to create a new project so that you can test different configurations as you read. Once you have your Azure DevOps project created, take a glance at your project settings and select the products that you’d like enabled.

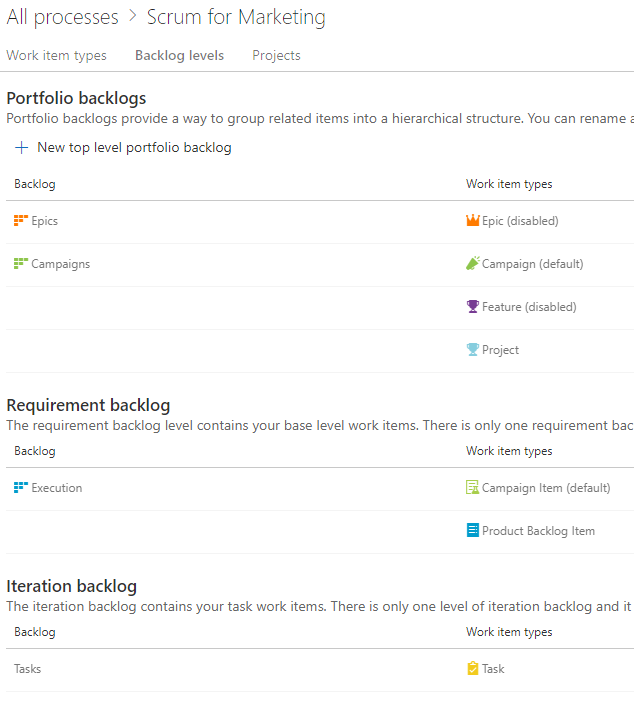
Below, you can see that I have all of the products enabled. For your team, you’ll want to equip them with the Visual Studio Enterprise subscription (formerly called MSDN Premium) so that they have licensing for all of the products. You’ll need them. Packages is the first one you’ll miss if you are using a free or lower license. And as we move through the book, you’ll make use of all the products in the Azure DevOps family.



*Figure: You can enable or disable any of the products in the Azure DevOps family.*

## Change your process template

The title of this section is not “choose your process template.” You will do that, but your organization has a workflow, and you must capture it and make the process template faithfully model that workflow. Within the first way of DevOps is the principle of “make work visible”. Azure Boards is the tool of choice for modeling the shape of your work. Azure Boards uses Work Items to track a unit of work. A work item can be of any type and has a status as well as any number of other fields you’d like. As you think about your hierarchy of work, don’t immediately start creating work items using the built-in sample hierarchy. Instead, think about the work that you already do and the parent-child relationships between some of the types of work. For example, in a marketing department, the structure in the figure below may be appropriate.



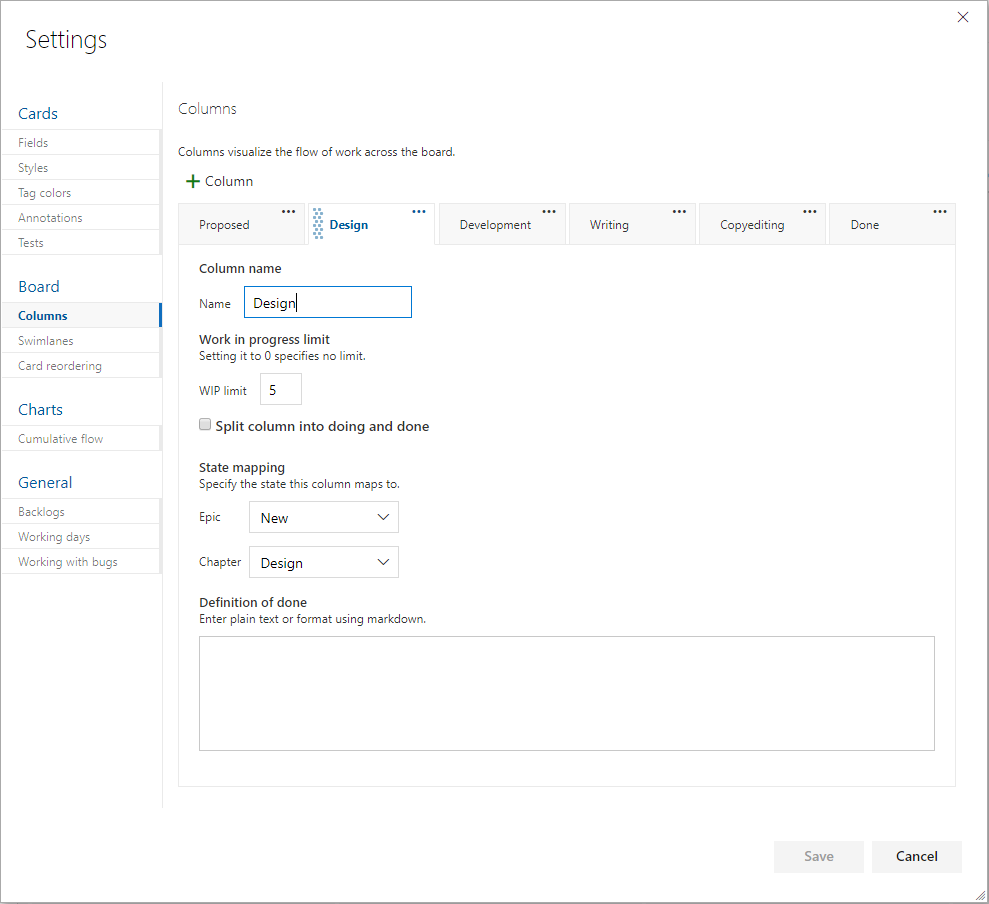
*Figure: A marketing department has Campaigns that are broken down into individual items.*

This marketing department has decided that they only need three levels of work. A Campaign can have multiple Campaign Items or Product Backlog Items. A Campaign Item and a Product Backlog Item can have multiple Tasks. At the top level, they can track at the Campaigns level or the Execution level. An individual iteration or sprint is tracked with Tasks. You can have any number of higher-level portfolio backlogs if you need higher levels of groupings. Even while the built-in process template includes **Epic > Feature > Product Backlog Item**, you’ll quickly outgrow this because it won’t match your organization. You need to disable most of the built-in work item types and create your own so that you can name them and put only the fields and the progression of statuses that make sense in your teams’ environments.

You may think of the following work types to get the creative juices flowing in order to capture the model of your organization’s world. Notice that I didn’t say “design the model.” Your model already exists. You need to capture the nouns and the verbs of your existing reality and make Azure Boards represent what’s already there. If you capture the wrong model, it won’t fit, and your co-workers will have a hard time tracking their work because it just won’t make sense. So, consider the following types:

* Business initiatives
* Marketable features
* Plannable work to budget, schedule, and fund
* Individual tasks to get done

This becomes the foundation of your usage of Azure Boards going forward. You’d never think of starting a new application with the Northwind or AdventureWorks database schema. Those tables were chosen by someone else. That model just doesn’t fit the nature of the data you’re trying to store. In this same way, the schema of the built-in process templates won’t fit your organization. You need to load your own model. Once you have your model, you need to specify the process of each major entity (work item). For example, if you were writing an article or a book, you might create a Chapter work item and specify the status progression on the Kanban board like that shown below.



*Figure: The columns all map to a state of a work item, and each can be assigned a definition of Done.*

By determining ahead of time what the process is to take a certain level of work item from creation to done, you organize your team. Each state, or board lane, should be owned by a type of role. For example, if you have a stakeholder designated as the person who’ll give the go ahead on the sketch of a screen before it’s developed, that stakeholder should have a column where they own the work within it. Each work item is represented by a visual card on the Kanban board, and the cards in their column are theirs to work. If the stakeholder does nothing, cards pile up in that column, and nothing is developed because of the bottleneck in that column. A dashboard report can bring this to light on a daily basis so that no column has too much work in it. The stakeholder’s job would be to either approve the sketch of the screen or initiate a conversation to fix it. In no case would you want a bad screen to be coded. That would be worse. By creating a good number of columns, mapped to the states of the work item, you can move the work through a known process where every column has a type of role responsible for performing a known set of work and then forwarding the work in process WIP to the next column. From a quality control perspective, every person starting on work has the obligation for inspecting the WIP to see if the work is ready for them yet. If something is missing, you stop the line and get it corrected before propagating the error further downstream.

For the purposes of software teams, the level of backlog that is prepopulated with Product Backlog Items in the case of the Scrum process template or User Stories in the case of the Agile process template, is the appropriate level for doing branches and pull requests as well as designed test cases, as you’ll see a bit later in the article. Iterations or sprints can be planned with work items from this level. Then, tasks can be organically created, completed or destroyed day by day. It’s often good to make plans based on the lowest backlog level and then break those down into tasks as needed on an ad-hoc basis during the sprint.

## Types of work items

Depending on the process template you choose when you create your project, you’ll start with a pre-defined set of work item types, statuses and swimlanes in your boards. You should change these because there are only three process templates built-in, and they are all very basic. Don’t expect to use them without customization except for very simple projects. You have three process templates to choose from when starting a project. If you have already created a project, and you want to choose a different process template, you are out of luck. Create a new project. If that ship sailed long ago, don’t fret. You can morph any of the project templates into just about anything you want.

The choices for project template are CMMI, Agile, and Scrum. The scrum template is probably the most widely used at the writing of this book, and it is the template that is maintained the most. But the Basic template is new, and has been simplified down to just the basics. If you don’t know what process template to use, and you don’t know the difference between these, **choose the Basic template**[[1]](#footnote-1) and modify it from there. It has the least pre-specified, and you’ll be able to add anything you like.

You will see some similarities and differences in the built-in process templates, but they all share more than they differ. The following table illustrates the configurations of the templates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Backlogs | CMMI | Agile | Scrum | Basic |
| Portfolio | Epic Feature | Epic Feature | Epic Feature | Epic |
| Requirements | Requirement | User Story | Product Backlog Item | Issue |
| Iteration | Task | Task | Task | Task |
| Others | Bug Issue Change Request Review Risk | Bug Issue | Bug Impediment | n/a |

*Table: Built-in process templates come with a set of work item types that are meant to be customized*

You can see how similar the process templates are, and you should examine each one to gain some ideas because each work item is configured with a certain number of fields, and the fields of each is likely not going to fit your needs. As with a database schema if you go forward with tables and columns that are not used, your data set ends up with many null values. This causes confusion with reporting. If you are not going to use a field, customize your template and remove it or hide it from a work item. Simple is better.

You may think that the above processes are so similar that it doesn’t matter which one you start with, but the Requirements level work item type will probably help you make your decision. Here are the fields in this key work item type out of the box.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CMMI | Agile | Scrum | Basic |
| Name | Requirement | User Story | Product Backlog Item | Issue |
| Main Section | Description (multi-line text) | Description (multi-line text) | Description (multi-line text) | Description (multi-line text) |
| Secondary Section | Impact Assessment (multi-line text) | Acceptance Criteria (multi-line text) | Acceptance Criteria (multi-line text) | n/a |
| Development Section | Development (links) | Development (links) | Development (links) | Development (links) |
| Related Work Section | Related Work (links) | Related Work (links) | Related Work (links) | Related Work (links) |
| Planning Section | Planning • Size  • Priority  • Triage  • Blocked  • Committed | Planning • Story Points  • Priority  • Risk | Details  • Priority  • Effort  • Business Value  • Value area | Planning   * Priority * Effort |
| Classification Section | Classification • Type  • Value area | Classification   * Value area | n/a |  |
| Other | Effort   * Original Estimate   Schedule   * Start Date * Finish Date   Build and test   * Integrated In * User Acceptance Test   Subject Matter Experts   * Subject matter expert 1 * Subject matter expert 2 * Subject matter expert 3 | n/a | n/a |  |

*Table: Structure of the main work item type per process template*

As you can see, the process templates start to diverge at this point. You can hide fields of the built-in work item types, but you can’t remove them. It’s a cleaner work tracking data model to add custom fields rather than hide most of the built-in fields.

## Customizing your process

With Azure DevOps, as with any project management tool, you can customize the states of the work. The task before you here is to make sure to model all of the states the work needs to go through in order to be finished. Many tools provide a board similar to this.

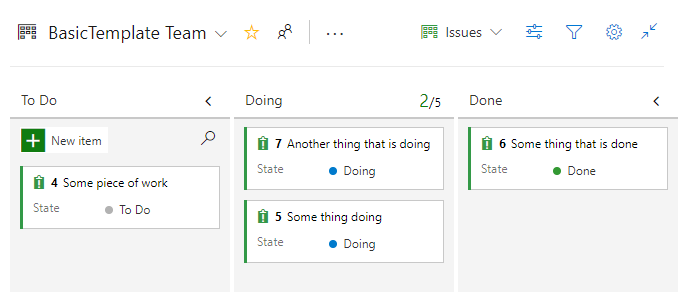


Figure: Each board starts out with simple states

This process is fine with a to-do list around the house, but it will not support any kind of serious software development project. There is much more activity inherent in the software development lifecycle. Regardless if your team would like to use Scrum or Kanban, or your some other methodology, you will need to decide on the unique states that work can be in at any given time. Here is a very common list of states you might choose. Each of which would show up as a column on your board. Note that I use the generic term “card” in place of Work Item, Issue, Product Backlog Item, User Story, etc. When modeling your board, each item of work will be manifested as a digital index card on a digital board. We suggest the typical owner of the card while it is in the corresponding state. Note that for the most part, the ownership cycles back and forth between product management and engineering. We are not being any more specific than that regarding roles. For teams who have adopted the DevOps ways as outlined in The Phoenix Project[[2]](#footnote-2), there are only those who commission the work and those who deliver the work. Any more organization beyond that is up to the team, which is consistent with the Manifesto for Agile Software Development[[3]](#footnote-3).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State | Activity | Done Criteria | Defects Types | Typical Owner |
| Idea/Backlog | Envision a software capability. Come up with and track the idea while contemplating priority. Create a card(work item) on the board. | An idea is deemed to be worth pursuing. | n/a | Product Management |
| Definition | Analyze the needed software behavior and capability and determine the user’s experience when exercising this new capability. Add this information to the card. | The software capability can be fully described from a user’s perspective. | Card is hard to describe. What the software should do in certain cases is unknown. Concept is not concrete enough to be workable | Product Management |
| Design | Make technical decisions needed to enable this capability. Diagrams, wire-frames, library, technology selection, logging levels, telemetry, etc. Break up the card into multiple cards of the right size for the team. | All key decisions on how to implement new capability have been determined. Cards are at the right size to proceed. | Design activity uncovers analysis gaps. Card’s scope is not concrete or understood. | Engineering |
| Test Spec | With a good understanding of the new capability, bullet list out the steps one would execute in order to test this card once it has been developed (the best format for acceptance criteria) | Card has test scenarios listed in a concrete, understandable fashion. | Unclear test procedures/ interface. Behavior combinations too many to simplify. | Product Management |
| Implement | Coding, configuring, building not only the functionality but also automating the tests specified for the card. | The card is working in a development environment, and all test scenarios are passing in the TDD/Development environment | Missing test scenarios or technology choices. Any gap in clear understanding of what is to be implemented. | Engineering |
| Inspect | Peer review, often a pull request from the feature branch. | The inspector has verified that the card’s implementation meets all items on the team’s inspection checklist. | Card doesn’t pass inspection checklist. | Engineering |
| Test | Product managers, stakeholders, users try out the implementation, manually running the test scenarios specified for the card. | All tests pass, and software executes without any new problems | Software build throws errors. Software doesn’t pass test scenarios specified for the card. | Product Management |
| Stabilize | Perform exploratory, edge case testing. Try to break it. Validate logging & telemetry. | No team members can fine problems. | A problem (defect) is found. | Engineering, Product Management |
| Release | Deploy card’s release candidate to production (or enable feature flag) | First use of card is verified in the logs. | New capability does not come online as expected. | Product Management |
| Done | First use of card is verified in the logs. | n/a | n/a | n/a |

Table: Representative board structure for common states in a software development project

The above might seem like too many states if you are working by yourself, but by the time you have three or more developers on your team, you’ll be glad you are able to see where the work is. Without this structure, you will forget what is holding up each card of work. Let’s simplify these states so that we can see a different view.

Plan

* Idea/Backlog
* Definition
* Design
* Test Spec

Build

* Implement

Check

* Inspect
* Test
* Stabilize
* Release

Interestingly, we are sandwiching the state that represents Building with four states on either side. If you just right into building without proper planning, there are four categories of decisions that will trip you up.

1. Idea: faulty concept of what to build
2. Definition: analysis gaps or unclear scope of what to build
3. Design: technology/architecture/pattern decisions needed in order to build
4. Test Spec: how to know when you’ve build everything you need to build

Again, if you are building the software as a team of one or two, you can simplify this down because you communicate frequently and take care of these things as they come. For teams larger than that, you need these concepts in some form, regardless of what you decide to name the states or columns.

## Working with the process

Now that you have determined for your organization how many stages, or swim lanes, are appropriate, you’ll need to integrate your version control system with Azure Boards in order to be able to track every code or asset change that is associated with a card (or Work Item, within Azure Boards).

While organizing our version control system is covered later in the book, we will now cover the basic things to do in order to integrate those changes. GitHub, acquired by Microsoft[[4]](#footnote-4) in 2018, is strategically meant to be the premium Git source control offering for Microsoft going forward. The work is happening this year to enable that: Microsoft AD sign-in, automatic pull request linking, etc. If your code is already inside GitHub, you can do some linking today at the time of writing. Inside your Azure DevOps project settings, you can connect your GitHub account.

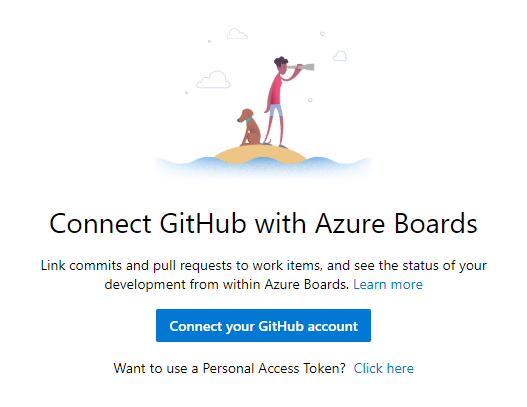


Figure: Navigate to Project Settings->Boards->GitHub connections, in order to begin the process.

### Linking commits

If you are not already a GitHub user, you can work with git source control right within Azure Repos. In the figure below, we are performing a commit right from within Visual Studio to our Azure Repos git repository.

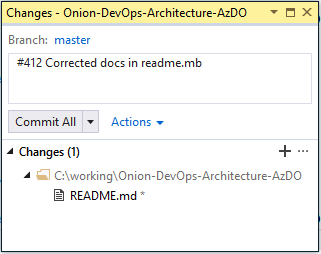


Figure: A commit from Visual Studio can auto-link with the work item by including #{work item id}

All it takes to link a commit with your work items in Azure Boards is to start the commit message with the work item number. You should do this every time, even if you are working by yourself. Along with better traceability on what changes a work item required, it will encourage the team to control scope and stay on track by only making changes for the work item in front of them.

SIDEBAR

As a developer, it is easy to get distracted when browsing code because you will see refactoring opportunities, and you will want to make the code better. It is not uncommon to look up after an hour of coding work to realize that you aren’t even working on the item you set out to complete. By setting a team rule that you will always link commits with a work item, you can keep yourself and your teammates on the most productive path every day.

Regardless of the git tool you elect to use, starting your commit message with the number of the current work item will cause Azure Boards to make the link.

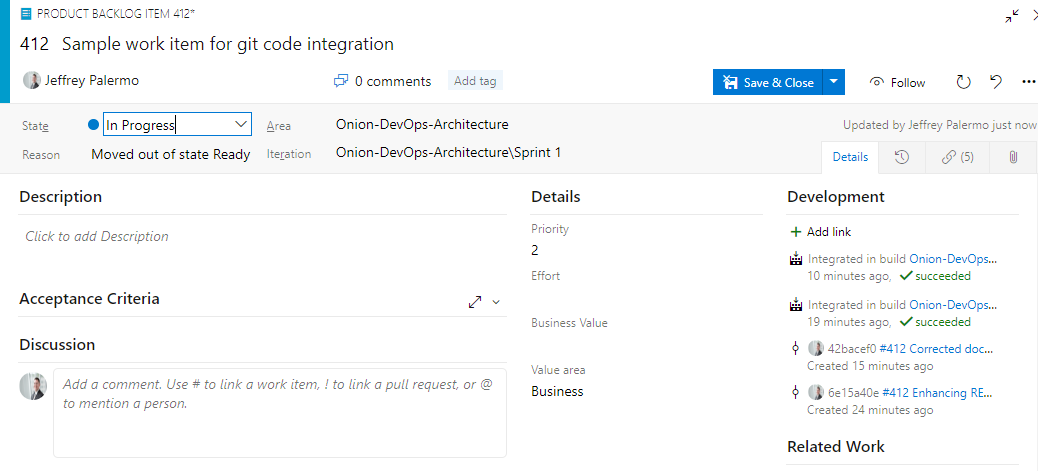


Figure: Azure Boards automatically links work items with git commits and builds that are related.

In the figure above, you can see that work item #412 has been linked with two git commits and the two resulting builds that contained these commits. If you tag your commits but don’t see this automatically, check in the Azure Repos settings that a teammate hasn’t turned this off. It is on by default for new projects but may not be on by default for repositories that were imported from outside sources. In that case, you’ll need to enable it for the new repository.

### Branching from Azure Boards

When working with work items of any type in Azure Boards. you'll want to build them up throughout the process. As you analyze the needed change and create screen mockups or any other document, either attach the document or include it inside the git repository itself. When you are ready to begin coding, create a branch. If you know how to organize your branches, then you are ahead of the game. If you are wondering what branching strategy to choose, then keep it simple and use plain feature branches instead of a “features/“ namespace. For more research on available branching strategies, see the Branch Organization docs from Microsoft[[5]](#footnote-5).

When starting to code on a work item, let Azure Boards do the work for you. In the following figure, you can see that we can create our feature branch right from the board.

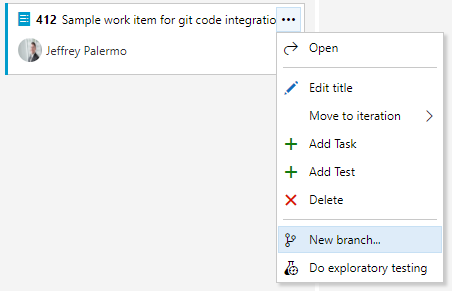


Figure: By clicking on the menu icon, we can create a new branch for development on the work item.

From here, we’ll want to maintain our team’s branching convention.

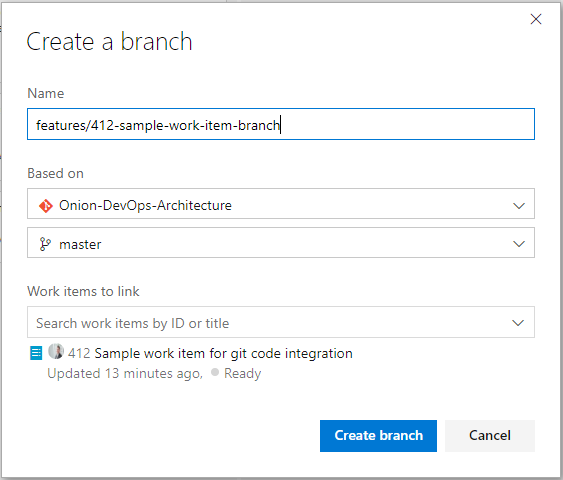


Figure: Place the work item number in the branch name to keep them organized.

You can create any branching scheme that you can imagine, so keep it simple. Place the work item number at the beginning so that you’ll be able to find your branch. Then, use all lower case with dashes. You can’t use spaces in a branch name. An added benefit of including the work item number in your branch name is that it will be a **constant reminder to stay on track** and only make changes that are needed for the current work in front of you.

Now that you have a branch for your work item, go to Visual Studio and check out the new branch from Team Explorer.

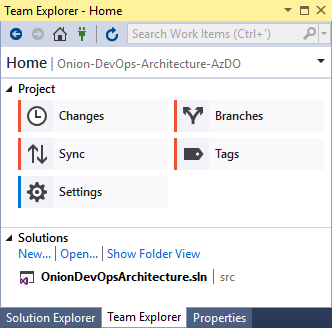


Figure: View the branches using Team Explorer.

Any commits you make with then stay on the new branch. If you break down features into user stories and tasks, remember to tag the commit message to the most specific work item you are working on. For record-keeping, you’ll have a branch that corresponds to your Feature (or Issue, or User Story), and then you’ll have individual commits tagged to it or any of the children that you’ve worked on.

When you have completed the work and are ready to merge your branch back in, you can create a pull request.

### Merging using pull requests

Within Azure DevOps, there are a few places where you can create a pull request. And while you can perform a git merge without a pull request, using one allows your team to integrate a formal inspection process, which is a proven way to find and prevent defects from being shipped to your customers.

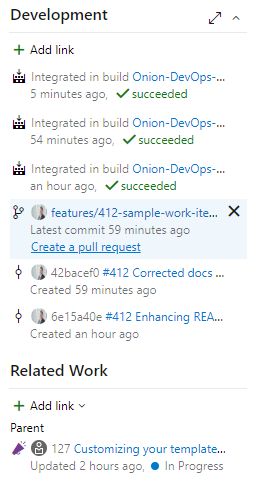


Figure: Within the work item screen, you can create a pull request.

In this figure, we can see the branch that was created for the work item. When we are finished building the feature that the work item represents, we can create a pull request so that our team can bring in the changes back to the master branch.

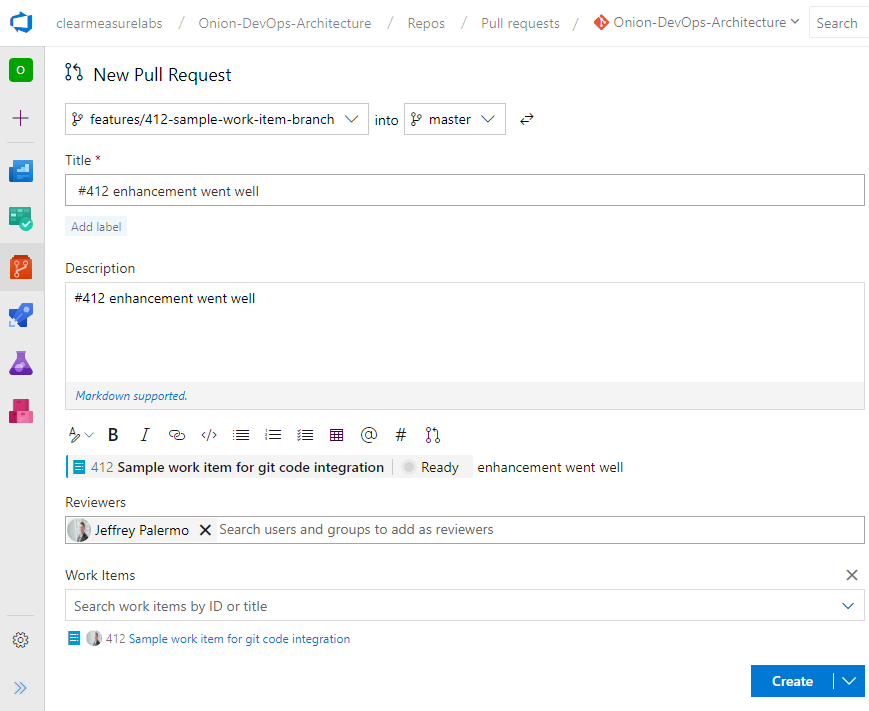


Figure: You can choose individuals or groups to be pull request reviewers.

After you’ve created your pull request, your team will be notified. They will be able to see and browse the changes you are bringing in. They will be able to comment on the changes and even have a back-and-forth conversation if necessary. This provides an opportunity to make any changes before the code is merged into master. If any changes are needed, simply make the changes on the branch. The pull request will update itself automatically. The pull request operates at the branch level, not the commit level. Therefore, if you need to do more work and make more commits, your pull request will not be invalidated. When approved, you can complete your pull request and monitor the automated merge.

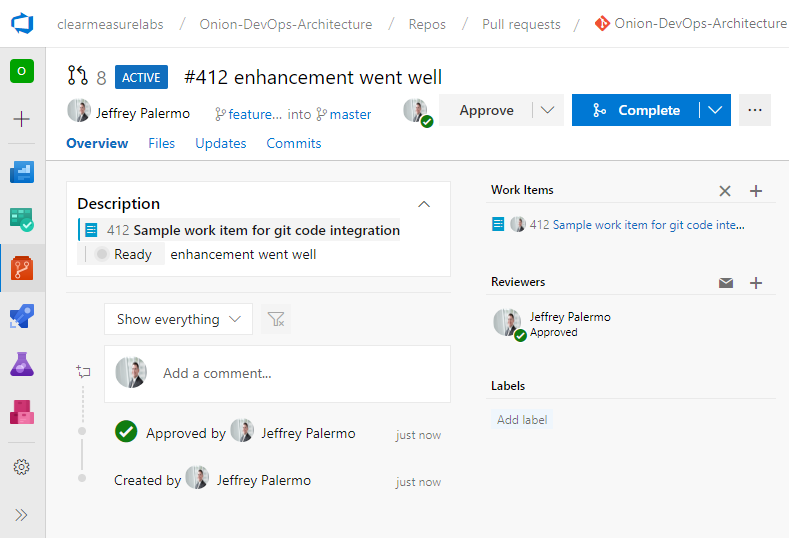


Figure: Complete the pull request after approvers have marked it as approved.

## Wrap up

When using Azure Boards to manage your software project, you benefit from the automatic integration with the rest of the Azure DevOps family. This chapter, while showing some fantastic capabilities, only scratches the surface on the power of Azure Boards. The purpose of this book is not to be a comprehensive feature guide for Azure DevOps. For more reading on Azure Boards, visit the official documentation[[6]](#footnote-6).

We have taken you through a micro-workflow of customizing your board and working with a software change through your customized process. This figure shows the level of details captured in just this small example. Your work items will be even richer with information as you track your work through your board.

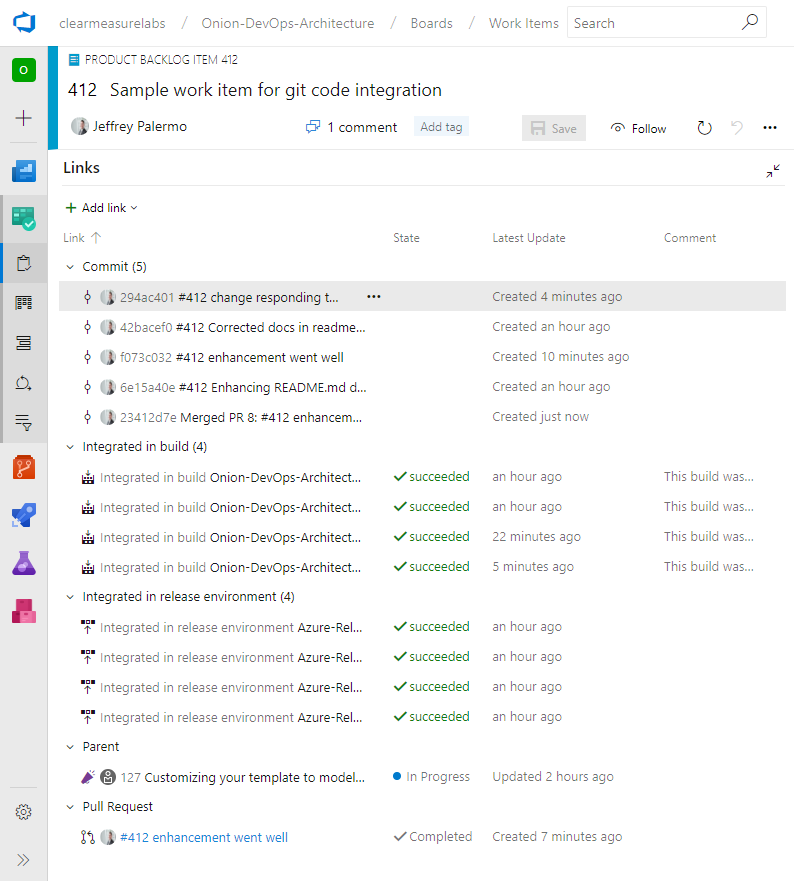


Figure: Your work item will become rich with information just by tracking it on Azure Boards.

Armed with this tool, you have a world-class project tracking capability, enabling you to focus on your code. In the next chapter, we’ll look at Azure Repos, but more importantly, how to set up your git repository for success.

# Bibliography

*Adopt a Git branching strategy*. (n.d.). Retrieved 2 18, 2019, from https://docs.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops

*Azure Boards Documentation*. (n.d.). Retrieved from https://docs.microsoft.com/en-us/azure/devops/boards/?view=azure-devops

Beck, K., Grenning, J., Martin, R. C., Beedle, M., Highsmith, J., Mellor, S., . . . Marick, B. (2001). *Manifesto for Agile Software Development*. Retrieved 2 18, 2019, from Agile Alliance: http://agilemanifesto.org/

Kim, G., Behr, K., & Spafford, G. (2013). *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. Retrieved 2 18, 2019, from https://amazon.com/phoenix-project-devops-helping-business/dp/0988262592

*Microsoft to acquire GitHub for $7.5 billion*. (n.d.). Retrieved 2 18, 2019, from https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/

1. Pit of success: start new process templates inheriting from Basic [↑](#footnote-ref-1)
2. (Kim, Behr, & Spafford, 2013) [↑](#footnote-ref-2)
3. (Beck, et al., 2001) [↑](#footnote-ref-3)
4. (Microsoft to acquire GitHub for $7.5 billion, n.d.) [↑](#footnote-ref-4)
5. (Adopt a Git branching strategy, n.d.) [↑](#footnote-ref-5)
6. (Azure Boards Documentation, n.d.) [↑](#footnote-ref-6)