Microsoft

Professional Scrum Development with Azure DevOps

Professional

Richard Hundhausen

Foreword by Ken Schwaber, Co-creator of Scrum

Praise for this book

"Scrum is described as taking 10 minutes to learn and a lifetime to master. In this book, Richard provides tips and tricks to mastering Scrum. He marries the practical with the abstract, providing a foundation of learning that helps Developers deliver high-value products and solve complex problems. If you are using Azure DevOps and want to get better at doing it, then this is the book for you."

—Dave West, Scrum.org Product Owner and CEO

"Like it or not, many teams need tooling to help them with their Scrum implementation. That's where Richard comes in. His knowledge and passion shine through in all that he touches—especially in this essential guide for how to use Azure DevOps for Scrum Teams. If you know anything about Richard, and you are using Azure DevOps with Scrum, then you'll know this book is a must-read."

—Daniel Vacanti, Co-founder, ActionableAgile

"In this book, Richard Hundhausen does a great job explaining and connecting the domains of Professional Scrum with professional development using Microsoft Azure DevOps. Richard introduces the history and current state in both domains and makes the book even richer with personal tips and illustrations through case studies."

—Gunther Verheyen, independent Scrum Caretaker,
Professional Scrum Trainer

"Scrum is a framework that is easy to understand but difficult to master. Richard takes the difficult out of the equation for you. What sets him apart from all others is his ability to help others not only understand Scrum, but become masters at it."

—Chris Roan, Wells Fargo Agile Transformation Leader

"If you're working on a Scrum team, do yourself a favor and read this book. In it, Richard distills his many years of practical experience leading Scrum teams in order to help you and your team accelerate your DevOps transformation. If you want to deliver more customer value at higher velocity, there's no better place to start."

—Jeff Beehler, Senior Director, Product Operations, GitHub, Inc.

"During my time on the Azure DevOps team, I became aware of Richard's passion for Professional Scrum and his desire for us to build the tool in a way that Scrum Teams would love it. The essence of DevOps is to get a right blend of processes, tools, and people working seamlessly to deliver customer value. Combine that with Scrum and you have a winner. Richard does a great job of taking the theory of Scrum and converting it into specific sets of actions that everyone in a team (Product Owners, Developers, Testers, stakeholders, etc.) can follow. If you want to be an expert at Scrum while putting it into day-to-day practice using Azure DevOps, this is the book for you!"

—Ravi Shanker, Principal Group Program Manager and former Product Owner for Azure Test Plans

"Richard successfully weaves three important concepts: Azure Devops, Scrum, and creating quality code. This book is a must-read for anyone interested in end-to-end solutioning within the Microsoft development environment."

—Donis Marshall, Microsoft MVP, Professional Scrum Developer, President of Innovation in Software

"Richard has been at the forefront of agile and Scrum since the beginning and was the first ALM/DevOps MVP. The book shows his vast knowledge and understanding of Professional Scrum and Azure DevOps. It's a must-have for teams to continue on their improvement journey."

—Philip Japikse, CTO Pintas & Mullins, Microsoft MVP, Professional Scrum Trainer

"Scrum is simple—or it seems that way until you actually try to implement it. The great thing about Richard's book is that it gives readers practical implementation advice to translate the simple words in the Scrum Guide into valuable actions by their teams."

—Steve Porter, Scrum.org Professional Series Manager, Professional Scrum Trainer

"Azure DevOps is a suite of tools and Scrum is a framework used to deliver a product in an iterative and incremental way. Both have a lot in common but are totally different beasts. Richard blends them together in a surprisingly delightful and easy-to-digest way that clearly explains how and where to apply both to help teams deliver better and more valuable software together."

—Jesse Houwing, Lead Consultant at Xpirit, Professional Scrum Trainer, Microsoft MVP

"If you are working with Azure DevOps, then this book should be required reading for everyone who touches that system. With 80 percent of all development teams using Scrum, this is the book to help you do Professional Scrum within Azure DevOps and improve the likelihood of success of your team."

—Martin Hinshelwod, naked Agility, Professional Scrum Trainer, Azure DevOps MVP

"Scrum is simple to understand but extremely hard to implement well. In his book Richard offers battle-hardened experience and practices with which mastering Scrum becomes achievable using Azure DevOps."

—Ognjen Bajic, Professional Scrum Trainer, Azure DevOps MVP

"Richard draws on his expertise as a Professional Scrum Trainer and DevOps guru to write a fantastic book that describes the "why" and the "how" of doing Scrum with Azure DevOps. Clear and well written, this should be on your bookshelf if you are using Azure DevOps with Scrum."

—Simon Reindl, Professional Scrum Trainer

"Just by following Richard's actionable recommendations, even first-time users will get a properly configured and productive implementation of Professional Scrum with Azure DevOps. Experienced practitioners will be able to confront their ways of work and experiences with real-world advice from the book."

—Ana Roje Ivancic, Professional Scrum Trainer, Azure DevOps MVP

"I use Azure DevOps every day. I did not realize how much I had to learn until I read *Professional Scrum Development with Azure DevOps*. It is filled with expert guidance for maximizing value for your team!"

—Cory Isakson, Microsoft Senior Consultant

"If you have to read one book about Scrum, make it the one you are holding right now. This book teaches you everything you need to know about Professional Scrum Development—clear and concise, without the fluff."

—Martin Kulov, Microsoft DevOps MVP, Microsoft Regional Director

This page intentionally left blank

Microsoft

Professional Scrum Development with Azure DevOps

Richard Hundhausen

PROFESSIONAL SCRUM DEVELOPMENT WITH AZURE DEVOPS
Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

ISBN-13: 978-0-13-678923-9
ISBN-10: 0-13-678923-4

Library of Congress Control Number: 2021930493

ScoutAutomatedPrintCode

CREDITS

Editor-in-Chief
Brett Bartow

Executive Editor
Loretta Yates

Sponsoring Editor
Charvi Arora

Development Editor
Songlin Qiu

Managing Editor
Sandra Schroeder

Senior Project Editor
Tracey Croom

Copy Editor
Liz Welch

Indexer
Tim Wright

Proofreader
Donna Mulder

Technical Editor
Donis Marshall

Editorial Assistant
Cindy Teeters

Cover Designer
Twist Creative, Seattle

Compositor
codeMantra

This book is dedicated to my Scrum Team:
Esmay, Isla, Berlin, Blaize, Sawyer, and Kristen.

—Richard Hundhausen

This page intentionally left blank

Contents at a Glance

This page intentionally left blank

Contents

About the Author

RICHARD HUNDHAUSEN is the president of Accentient, a company that helps software organizations and teams deliver better products by understanding and leveraging Azure DevOps and Scrum. He is a Professional Scrum Trainer and co-creator of the Nexus Scaled Scrum framework.

As a software developer, consultant, and trainer with nearly 40 years of experience, he understands that software is built and delivered by people and not by processes or tools. You can reach Richard at *richard@accentient.com*.

Foreword

By 2001, the software industry was in trouble—more projects were failing than succeeding. Customers began demanding contracts with penalties and sending work offshore. Some software developers, though, had increasing success with a development process known as "lightweight." Almost uniformly, these processes were based on the well-known iterative, incremental process.

In February 2001, these developers issued a manifesto—the Agile Manifesto. The Manifesto called for Agile software development based on four principle values and twelve underlying principles. Two of the principles were 1) to satisfy customers through early and continuous delivery of working software, and 2) to deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.

By 2009, the Scrum Agile process was used predominantly. A simple framework, it provided an easily adopted iterative incremental framework for software development. It also incorporated the Agile Manifesto's values and principles. The two authors of Scrum, Jeff Sutherland and myself, also were among the authors of the Agile Manifesto.

I had anticipated some of the difficulties organizations (and even teams) would face when they adopted Scrum. However, I believed that developers would bloom in a Scrum environment. Stifled and choked by waterfall, developers would stand tall, employing development practices, collaboration, and tooling that nobody had time to use in waterfall projects.

Much to my surprise, this was only true for perhaps 20 percent of all software developers.

In 2009, Martin Fowler characterized most Agile software development as "flaccid":

There's a mess I've heard about with quite a few projects recently.
It works out like this:

- They want to use an Agile process, and pick Scrum.
- They adopt the Scrum practices, and maybe even the principles.
- After a while, progress is slow because the codebase is a mess.

What's happened is that they haven't paid enough attention to the internal quality of their software. If you make that mistake you'll soon find your productivity dragged down because it's much harder to add new features than you'd like. You've taken on a crippling Technical Debt *and your Scrum has gone weak at the knees. (And if you've been in a real scrum, you'll know that's a Bad Thing.)" http://martinfowler.com/bliki/FlaccidScrum.html*

Martin's description of flaccid Scrum resonated with our experience. Most developers were skilled, but not adequately skilled in the three dimensions required to rapidly build complete increments of usable functionality. These dimensions are:

- **People** The ability to work in a small, cross-functional, self-managing team.
- **Practices** The knowledge of and ability to apply modern engineering practices that short cycle development mandates.
- **Tooling** Tools that integrate and automate these practices so that successive increments can be rapidly integrated without the drag of exponentially accruing artifacts that must be handled manually.

We put our business on hold while we worked through 2009 to create what has become known as the Professional Scrum Developer program. Offered in a three-day format, we formulated a workshop. The input was developers whose knowledge and capabilities produced flaccid increments. The output were teams of developers who had developed solid increments of software called for by the Agile Manifesto and demanded by the modern, competitive organization.

Richard has been there since the beginning. His book, *Professional Scrum Development with Azure DevOps*, continues his participation in the movement started by us few in 2009.

When you read Richard's book, you can learn the three dimensions needed for Agile software development: people, practices, and tools. Just like in the course, Richard intertwines them into something you can absorb. If you are on a Scrum team, read Richard's book. List the called-for practices. Identify which practices pose challenges to your team. Order them by their greatest impact. Then remediate them, one by one.

Many people spend money going to Agile conferences. Save the money and more by buying this book, discussing it with others, and going to meetups and code camps—the "un-conferences" for the serious.

Richard and I look forward to your increased skill. Our industry and our society need it. Software is the last great scalable resource needed by our increasingly complex society. The effective, productive teamwork of Agile teams is the basis of problem solving that our society also needs.

Scrum on!

Ken Schwaber

Co-creator of Scrum

Introduction

Scrum is a framework for developing and sustaining complex products such as software. Scrum is just a set of rules, as defined in the *Scrum Guide* (*https://scrumguides.org*), and it describes the roles, events, and artifacts, as well as the rules that bind them together. When used correctly, this framework enables a team to address complex problems while productively and creatively delivering products of the highest possible value. Scrum is an Agile method. In fact, it is the most popular Agile method in use today.

Scrum employs an iterative and incremental approach to optimizing predictability and controlling risk. This is due to the empirical process control nature of Scrum. Through proper use of inspection, adaptation, and transparency, a Scrum Team can try a new way of doing something (an experiment) and gauge its usefulness after a short iteration. They can then collectively decide to embrace, extend, or drop the practice. This includes the tools a team uses and how they use them.

Combining Scrum with the tools found in Microsoft Azure DevOps is a powerful marriage. It is the purpose of this book to establish a baseline understanding of Scrum and how Scrum is supported in Azure DevOps. I will also illustrate which practices provide more value when executed without the use of tools. In addition, I will point out the tools that have been erroneously marketed as agile and contrast them with more preferred practices.

In software development, anything and everything can change in a moment's notice. Healthy teams know this. They also know that continuously inspecting and adapting the way things are done is a way of life. High-performance Scrum Teams take this a step further. They know that within every impediment or dysfunction is an opportunity to learn and improve. Reading this book is a great first step.

This book primarily focuses on using Scrum for software products, mostly because that's the target domain for Azure DevOps. Much of this book, however, is applicable beyond software development and IT projects. Since Scrum is a lightweight framework for developing adaptive solutions for all types of complex problems, the guidance in this book can apply to developing any kind of product, such as a service, a physical product, or something more abstract.

This Book Might Not Be for You If . . .

This book is intended for teams using Scrum and Azure DevOps together as they develop complex products, such as software. It won't provide as much value for non-Scrum teams or Scrum teams developing products that are not complex. It won't provide any value for teams running formal waterfall or sequential software development projects, except to hopefully change the minds of such proponents. Likewise, if a team is using Scrum but not yet using Azure DevOps, the bulk of the book won't be very interesting, except to define and highlight Professional Scrum and point out what goodness those teams might be missing out on. This is also the case for teams using older versions of Team Foundation Server, which won't contain the latest, high-value, team-based tools for planning and managing work and enabling team collaboration.

If you are looking for "best practices," then you have the wrong book and the wrong author. I refuse to use that term because it implies a couple of wrong assumptions: (1) that this practice truly is "best" for all teams working on all products in all organizations, and (2) that a team can stop looking and experimenting once they've found that best practice. I prefer the term "proven practice" instead. Regardless of what you or I call it, this book is full of many practices for you and your team to consider on its improvement journey.

Organization of This Book

This book is divided into three sections, each of which focuses on a different aspect of the marriage of Professional Scrum and Azure DevOps. Part I, "Scrumdamentals," sets a baseline understanding of the Scrum framework, Professional Scrum, Azure DevOps, and specifically the Azure Boards service. Part II, "Practicing Professional Scrum," consists of several chapters detailing the practical application of how a Professional Scrum Team would use the relevant features of Azure DevOps to create and manage a Product Backlog, plan a Sprint, create a Sprint Backlog, and effectively collaborate during the Sprint. Part III, “Improving,” includes a chapter on defining and improving a Scrum Team’s flow, identifying common challenges and dysfunctions in order to remove them, and using techniques to continually improve your game of Scrum. There is also a chapter on how to improve at scale by adopting Scaled Professional Scrum using the Nexus scaled Scrum framework. By reading all sections sequentially, you will see how Azure DevOps and Scrum can be used together in an effective way and how a Scrum Team can evolve into a Professional Scrum Team and, further, into a high-performance Professional Scrum Team.

Throughout each chapter, I suggest and recommend many practices and patterns of working. I use terms like *Professional* Scrum Team and *high-performance* Scrum Team to differentiate from garden-variety Scrum Teams—those practicing mechanical Scrum without attention to inspection, adaptation, and improving. At times you may dismiss my guidance as “magical thinking” and assume that I don’t live in the real world. You may think that the ideas I propose won’t work for your team, with your people, in your organization. Although it’s true that I don’t know the specifics of your organization, I’m confident that improvement can be made regardless of the amount of friction you might face. I’ve seen it and hundreds of my Professional Scrum Trainer colleagues have, too. Keep in mind that my descriptions of these high-performance behaviors should be considered a vision or “perfection goal” of what your team can achieve. It will be hard. It will take time. It will take help. Ultimately it will be people like you who lead the improvement journey.

Finding Your Best Starting Point in This Book

The different sections of *Professional Scrum Development with Azure DevOps* cover a range of topics. Depending on your needs and your existing understanding of Scrum, Azure DevOps, and the related practices, you may wish to focus on specific areas of the book. Use the following table to determine how best to proceed through the book.

If you are	Follow these steps
New to Scrum or have never heard of it	Read the *Scrum Guide* and then read Chapter 1
New to Professional Scrum or have never heard of it	Read Chapter 1
New to Azure DevOps or its suite of tools	Read Chapter 2
New to the Azure Boards service or want to know how to create a custom, Professional Scrum process	Read Chapter 3
Familiar with Scrum and Azure DevOps and only want to learn how to set up Azure DevOps for a Scrum Team	Read Chapter 4

If you are	Follow these steps
Familiar with Scrum and Azure DevOps and only want to learn how to plan a Sprint and create a Sprint Backlog	Read Chapter 6
New to the concept of acceptance test-driven development and how to plan and track a Sprint using Azure Test Plans	Read Chapter 7
New to the concept of flow or how a Scrum Team can use the Kanban board to visualize work and manage its flow	Read Chapter 9
Facing common Scrum challenges and are interested in overcoming dysfunctional behavior	Read Chapter 10
Facing a scaling situation where several Scrum Teams are collaborating to build a common product	Read Chapter 11

Conventions and Features in This Book

This book presents information using conventions designed to make the information readable and easy to follow.

- Screenshots from relevant Azure DevOps features are provided for your reference.
- Boxed elements with labels such as "Note" or "Tip" provide additional information and guidance related to the subject.
- Some notes and tips are practical guidance provided by fellow Professional Scrum Developers and Professional Scrum Trainers who have helped review this book.

In addition, I have included two additional boxed elements, one labeled "Smells" and the other labeled "Fabrikam Fiber Case Study."

Smell Throughout this book, I point out specific situations and traps that a Scrum Team or its members should avoid. I refer to these as *smells*. These smells typically—but not always—indicate an underlying dysfunction or other unhealthy behavior. For teams new to Scrum, these smells may be hard to identify. Once they are brought to light, however, they should be mitigated and used as learning opportunities. As a team improves, it should be able to recognize dysfunction on its own, as well as remove it. Professional Scrum Teams have the ability to identify potential waste or dysfunction, evaluate the risks, and even decide to opt in to specific behaviors, including those that may be a smell to the uneducated.

Fabrikam Fiber Case Study

As you flip through the pages, you will read about Fabrikam Fiber as our case study. Fabrikam Fiber is a fictional broadband communications provider (think: Cox, Sparklight, Charter/Spectrum, Comcast, etc.). Fabrikam Fiber is a large corporation that provides services for multiple U.S. states. They also use an on-premises web application for their customer service representatives to create and manage tickets for customer support issues. The team has been using Scrum for some time and has recently moved to Azure DevOps. My opinions on healthy and unhealthy behaviors are made evident through the choices made by the Fabrikam Fiber Scrum Team.

System Requirements

Although this book does not contain any hands-on exercises, I encourage you to sign up for Azure DevOps Services in order to experiment and learn as you read. It takes only a few minutes to create an organization, and the first five users are free on the Basic Plan—which is more than adequate for you and some colleagues to use all the features mentioned in this book. Azure DevOps Services is a cloud-based SaaS offering delivering new features every three weeks, which means that the screenshots in this book may not match what you see in your browser.

In addition, you may want to download Visual Studio Community Edition or Visual Studio Code to explore how they connect to Azure DevOps and how they can be used for collaboration using Azure Boards and Azure Repos. Both of these products are free.

Downloads: Code Samples

This book contains no code samples.

Acknowledgments

There are several people who helped me write this book. Thanks to: Loretta Yates for giving me another opportunity to write for Microsoft Press; Charvi Arora, Tracey Croom, Elizabeth Welch, Songlin Qiu, Vaishnavi Venkatesan and Donna Mulder for patiently reviewing my content and helping me get the styles right; Donis Marshall for inspiring me to write another book and giving me such direct (and valuable) feedback; Dan Hellem for

answering scores of Azure Boards questions and reviewing chapters; Phil Japikse, Simon Reindl, Brian Randell, Ognjen Bajić, Ana Roje Ivančić, Martin Kulov, Cory Isakson, David Corbin, Charles Revell, Daniel Vacanti, and Christian Hassa for providing some great ideas and helping me sharpen the message; and Ken Schwaber and Jeff Sutherland for updating the *Scrum Guide* after I was almost done writing this book. ☺

Errata, Updates, and Book Support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

MicrosoftPressStore.com/ProfScrumDevelopment/errata

If you discover an error that is not already listed, please submit it to us at the same page.

For additional book support and information, please visit *www.MicrosoftPressStore.com/Support*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

Stay in Touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*

This page intentionally left blank

CHAPTER 3

Azure Boards

As I mentioned in the previous chapter, Azure Boards is the Azure DevOps service that helps teams plan and track their work. It's the service that provides the work items, backlogs, boards, queries, and charts—all the building blocks that a team needs to visualize and manage their work.

The look and feel of Azure Boards is partially driven by the process that a team selects when they create the project. This process defines the building blocks of the work item tracking system. It also serves as the basis for any process model customization that a team might want to perform.

In this chapter I will dive into Azure Boards and discuss the various processes that can be selected, focusing on the Scrum process. I will also show you how to create an inherited process to customize Azure Boards' behavior. In Part II of this book, "Practicing Professional Scrum," I will delve even deeper into how the backlogs and boards explicitly support Scrum.

Choosing a Process

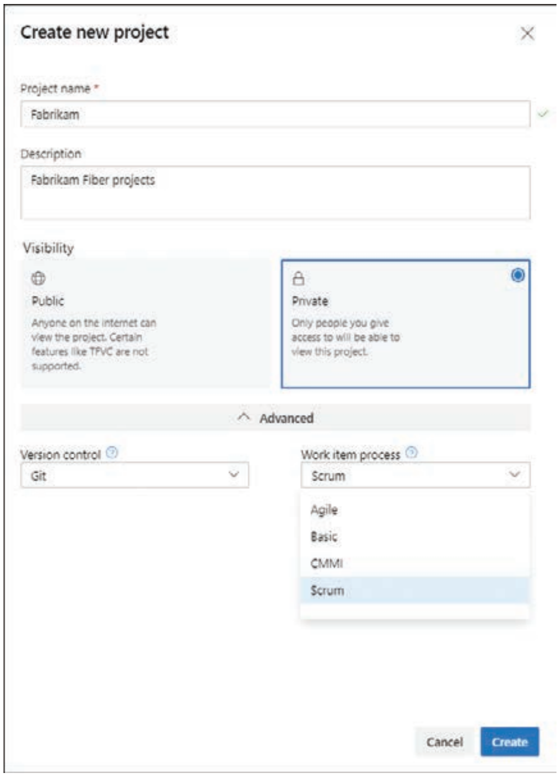
Several processes are available out of the box. These system processes are designed to meet the needs of most teams. Some of them are more formal, like the Capability Maturity Model Integration (CMMI) process. Some of them are lightweight, like the Basic process. Some of them are intended to match the *Scrum Guide*, like the Scrum process.

Here are the system processes available when creating a new project:

- **Agile** For teams that use agile planning methods, use user stories, and track development and test activities separately
- **Basic** For teams that want the simplest model that uses issues, tasks, and epics to track work
- **CMMI** For teams that follow more formal project methods that require a framework for process improvement and an auditable record of decisions
- **Scrum** For teams that practice Scrum and track Product Backlog items (PBIs) on the backlog and boards

These system processes differ mainly in the work item types that they provide for planning and tracking work. Basic is the most lightweight and closely matches GitHub's work item types. Scrum is the next most lightweight. The Agile process is a bit "heavier" but supports many agile method terms. CMMI provides the most support for formal processes and change management.

When creating a project, a process must be selected, as you can see in Figure 3-1. After creation, the project will use the work item types, workflow states, and backlog configurations as defined by that process.


Create new project

Project name *
Fabrikam

Description
Fabrikam Fiber projects

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

[✓ Private]
Only people you give access to will be able to view this project.

Advanced

Version control: Git

Work item process: Scrum

- Agile
- Basic
- CMMI
- Scrum

FIGURE 3-1 Selecting a process when creating a new project.

Note A *process* is different than a *process template*. A process defines the building blocks of the work item tracking system, supports the inheritance process model, and supports customization through a rich UI. It's available in Azure DevOps Services and Azure DevOps Server, but not for legacy Team Foundation Server versions. A process template is the legacy way of defining the building blocks of the work item tracking system. Process templates are expressed in XML and support customization through the modification and importing of XML definition files.

Work Item Types

Work items are the core elements of planning and tracking within Azure DevOps. They identify and describe requirements, tasks, bugs, test cases, and other concepts. Work items track what a team and team members have to do, as well as what they have done. Work items, and the metrics derived from them, can be visible within various queries, charts, dashboards, and analytics.

You can use work items to track anything that your team needs to track. Each work item represents an object stored in the work item data store. Each work item is based on a work item type and is assigned an identifier that is unique within an organization (or project collection in Azure DevOps Server). The work item types that are available to the project are based on the process used when the project was created, as you can see in Table 3-1.

TABLE 3-1 Work item categories available across the different processes.

	Scrum	**Agile**	**CMMI**	**Basic**
Work Item Category				
Requirement	Product Backlog Item	User Story	Requirement	Issue
Epic	Epic	Epic	Epic	Epic
Feature	Feature	Feature	Feature	-
Bug	Bug	Bug	Bug	Issue
Task	Task	Task	Task	Task
Test Case	Test Case	Test Case	Test Case	Test Case
Issue	Impediment	Issue	Issue	-
Change Request	-	-	Change Request	-
Review	-	-	Review	-
Risk	-	-	Risk	-

As you can see, the Agile process is very similar to the Scrum process. As far as work item types are concerned, the only differences are the type names of the *Requirement* and *Issue* work item categories. Agile refers to them as a *User Story* and *Issue*, respectively, whereas Scrum refers to them as a *Product Backlog Item* and *Impediment*, respectively. Figure 3-2 shows an example of this.

Note Microsoft introduced work item *categories* in Team Foundation Server 2010. Categories are essentially a meta-type and enable the various processes to have their own names and behaviors of work item types, without breaking the functionality of Azure Boards. Examples of work item categories that have different names include Requirement, Bug, and Issue.

You can also see how heavy and formal the CMMI process is, with official Change Request, Review, and Risk work item types, as well as the antiquated Requirement work item type. I've helped hundreds of teams install, understand, and use Azure DevOps and Team Foundation Server and can count the number of CMMI projects I've run into on one hand. Conversely, the Basic process has only a few work item types—just barely sufficient to track work and also to more closely match how work is managed on GitHub. It is also the default process, so there are many Basic process projects in existence, if only by accident.

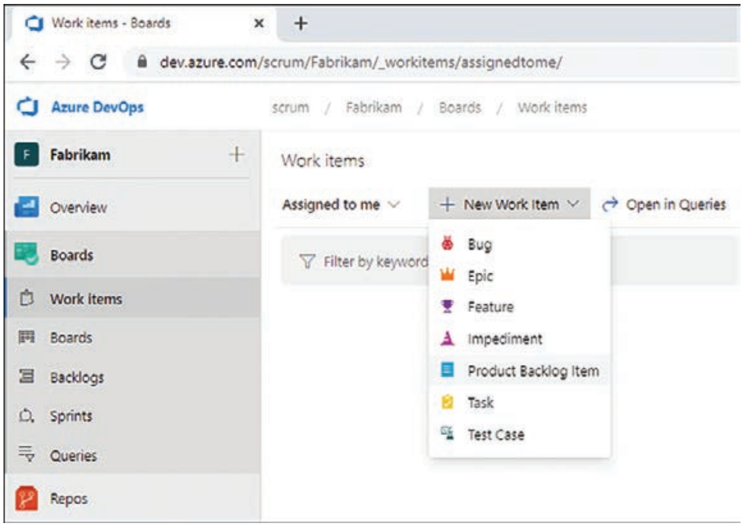


FIGURE 3-2 Work item types available to a Scrum project.

Another distinguishing feature of the different processes is the workflow states for the requirement category work item types. The workflow states define how a work item progresses upon its creation to its closure. You can see this natural progression by process in Table 3-2. Each state belongs to a state category (formerly known as a metastate). State categories enable the agile tools in Azure Boards to operate in a standard way regardless of the project's process.

TABLE 3-2 Requirement workflow states across the different processes.

Scrum	Agile	CMMI	Basic
New	New	Proposed	To Do
Approved	Active	Active	Doing
Committed	Resolved	Resolved	Done
Done	Closed	Closed	
Removed	Removed		

Hidden Work Item Types

Team Foundation Server 2012 introduced the concept of a *hidden* work item type. Work item types that are in this category are not able to be created from the standard user interfaces, such as the New Work Item drop-down list in Azure Boards. The reasoning behind this is that there are specialized tools for creating and managing these types of work items. Besides, creating these types of work items in an ad hoc way outside the context of the tooling doesn't make sense.

All processes, even the Basic one, support these hidden work item types:

- **Shared Parameter, Shared Steps, Test Plan** and **Test Suite** Created and managed by the tools in Azure Test Plans. I will take a closer look at all the testing work item types in Chapter 7, "Planning with Tests."

- **Feedback Request** and **Feedback Response** Used to request and respond to stakeholder feedback using the Test & Feedback extension.
- **Code Review Request** and **Code Review Response** Used to exchange messages in legacy Team Foundation Version Control (TFVC) code review in the My Work page in Visual Studio Team Explorer. These code reviews are not to be confused with those related to Git pull requests.

Microsoft knew that teams typically wouldn't be creating these work item types outside the context of their dedicated tools. They actually did us a favor by hiding them from the various UIs where we create and manage work items. Referring back to Figure 3-2, notice that there weren't any of these hidden work item types listed.

The Scrum Process

Shortly after Microsoft released Team Foundation Server 2010, they made the Microsoft Visual Studio Scrum version 1.0 process template available for download. This new template was designed from the ground up to embrace the rules of Scrum as defined in the *Scrum Guide*. It was the result of collaboration between Microsoft, Scrum.org, and the Professional Scrum community. Everyone knew that Scrum had become the dominant agile framework in software development. Microsoft recognized this as well. They also knew that teams using Team Foundation Server and Scrum together wanted a lighter-weight experience, resulting in less friction. What resulted was a minimalistic process template that followed the rules of Scrum. There were over 100,000 downloads of this new process template in the first couple of years.

Over the years, through ongoing collaboration with the Professional Scrum community, Microsoft learned a thing or two about the Scrum process and the community using it. Primarily, they have learned that teams liked it! These teams appreciate its simplicity and straightforward support of Scrum. As you saw in Table 3-1, there are not a lot of extraneous work item types beyond what is needed to plan and track a project using Scrum. In fact, it's even more lightweight than the Agile process.

Many Scrum Teams evaluating Azure Boards currently use whiteboards and sticky notes to track their work. Since you can't get any lighter weight than that, any prospective software tool would need to be as lightweight as possible. We kept this guiding principle in mind as we created the Scrum process, and I still keep it in mind as I write this book.

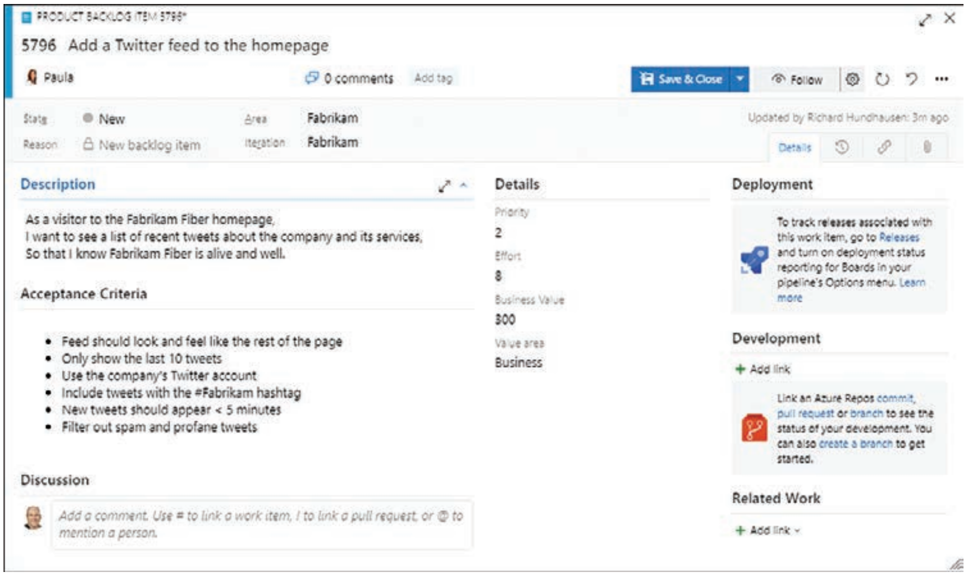
Scrum Work Item Types

I want to spend some time talking specifically about the work item types in the Scrum process, and how a Scrum Team should (and shouldn't) use them. I will focus on just those items that directly relate to planning and executing work. The work items related to Azure Test Plans (test plans, test suites, test cases, etc.) will be covered in Chapter 7.

Product Backlog Item

In Scrum, the Product Backlog is an ordered (prioritized) list of the outstanding work necessary to realize the vision of the product. This list can contain new things that don't exist yet (features), as well as broken things that need to be fixed (bugs). In Azure Boards, the Product Backlog Item (PBI) work item type enables the Scrum Team to capture all of these various requirements with the least amount of documentation as is necessary. In fact, only the *title* field is required.

Later, as more detail emerges, the PBI can be updated to include business value, acceptance criteria, and an estimation of effort, as you can see in Figure 3-3.


PRODUCT BACKLOG ITEM 5796*

5796 Add a Twitter feed to the homepage

Paula

0 comments Add tag

Save & Close Follow

State: New
Reason: New backlog item
Area: Fabrikam
Iteration: Fabrikam

Updated by Richard Hundhausen: 3m ago

Details

Description

As a visitor to the Fabrikam Fiber homepage,
I want to see a list of recent tweets about the company and its services,
So that I know Fabrikam Fiber is alive and well.

Acceptance Criteria

- Feed should look and feel like the rest of the page
- Only show the last 10 tweets
- Use the company's Twitter account
- Include tweets with the #Fabrikam hashtag
- New tweets should appear < 5 minutes
- Filter out spam and profane tweets

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

Details

Priority: 2
Effort: 8
Business Value: 300
Value area: Business

Deployment

To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development

+ Add link

Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

Related Work

+ Add link

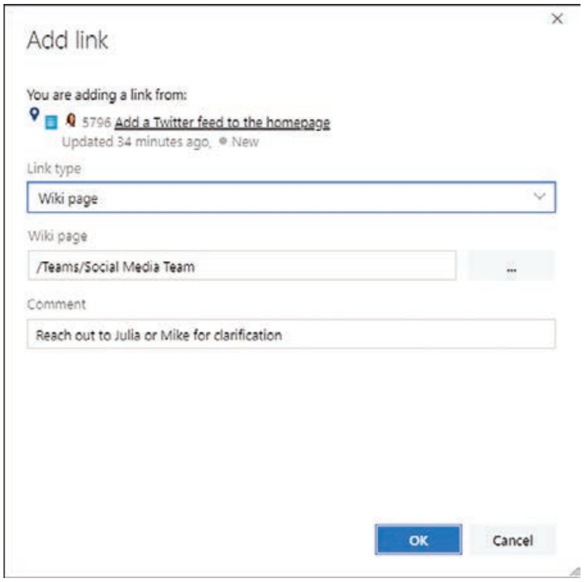
FIGURE 3-3 Adding detail to a PBI work item.

As you create or edit PBI work items, consider the following Professional Scrum guidance while entering data in the pertinent fields:

- **Title (Required)** Enter a short description that succinctly identifies the PBI.
- **Assigned To** Select the Product Owner or leave blank, but don't assign to a Developer. This will reinforce the fact that the whole team owns the work on the PBI.
- **Tags** Optionally add tag(s) to help find, filter, and identify the PBI. For example, some Scrum Teams opt not to use the Bug work item type in lieu of the PBI work item type and will simply tag those PBIs with "Bug."
- **State** Select the appropriate state of the PBI. States are covered later in this section.
- **Area** Select the best area path for the PBI. Areas must be set up ahead of time and can represent functional, logical, or physical areas or features of the product. If the PBI applies to all areas

your team covers or you aren't sure of the specific area, then leave it set to its default value. For Nexus implementations, each team within a project can have its own corresponding areas as well as a default area. I will talk about Nexus in Chapter 11, "Scaled Professional Scrum."

- **Iteration** Select the Sprint in which the Developers forecast that they will develop the PBI. If they have yet to forecast the PBI, then leave it set to the default (root) value.
- **Description** Provide as much detail as necessary so that another team member or stakeholder can understand the purpose of the PBI. The user story format (As a <type of user>, I want <some goal>, so that <some reason>) works well here to ensure that the who, what, and why are captured. You should avoid using this field as a repository for detailed requirements, specifications, or designs.
- **Acceptance Criteria** Describe the conditions that will be used to verify whether the team has developed the PBI according to expectations. Acceptance criteria should be clear, concise, and testable. You should avoid using this field as a repository for detailed requirements. Bulleted items work well. Gherkin (given-when-then) expressions work even better.
- **Discussion** Add or curate rich text comments relating to the PBI. You can mention someone, a group, a work item, or a pull request as you add a comment. Professional Scrum Teams prefer higher-fidelity, in-person communication instead.
- **Effort** Enter a number that indicates a relative rating (size) of the amount of work that will be required to develop the PBI. Larger numbers indicate more effort than smaller numbers. Fibonacci numbers (story points) work well here. T-shirt sizes (S, M, L, XL) don't, only because this is a numeric field. Effort can be considered the (I)nvestment in Return on Investment (ROI).
- **Business Value** Enter a number that indicates a fixed or relative value of delivering the PBI. Larger numbers indicate more value than smaller numbers. Fibonacci numbers work well here. Business Value can be considered the (R)eturn in ROI.
- **Links** Add a link to one or more work items or resources (build artifacts, code branches, commits, pull requests, tags, GitHub commits, GitHub issues, GitHub pull requests, test artifacts, wiki pages, hyperlinks, documents, and version-controlled items). You can see an example of linking a PBI to a wiki page in Figure 3-4. You should avoid explicitly linking PBIs to other PBIs, features, or epics using the Links tab. Instead, use drag and drop to establish hierarchical relationships within the backlogs. I will cover this in Chapter 5, "The Product Backlog."
- **Attachments** Attach one or more files that provide more details about the PBI. Some teams like to attach notes, whiteboard photos, or even audio/video recordings of the Product Backlog refinement sessions and Sprint Planning meetings.
- **History** Every time a team member updates the work item, Azure Boards tracks the team member who made the change and the fields that were changed. This tab displays a history of all those changes. The contents are read-only.


Add link

You are adding a link from:

5796 Add a Twitter feed to the homepage

Updated 34 minutes ago, ● New

Link type: Wiki page

Wiki page: /Teams/Social Media Team

Comment: Reach out to Julia or Mike for clarification

FIGURE 3-4 Adding a link to a wiki page.

While the PBI progresses on its journey to "ready" for Sprint Planning, the previous list of fields are really the only ones that need to be considered and completed. For the other fields on the PBI work item form, you should discuss as a team whether or not you should be using them because tracking data in those fields is most likely waste. When the PBI is forecast to be developed, additional fields and links will start to emerge, including links to task and test case work items, test results, commits, and builds.

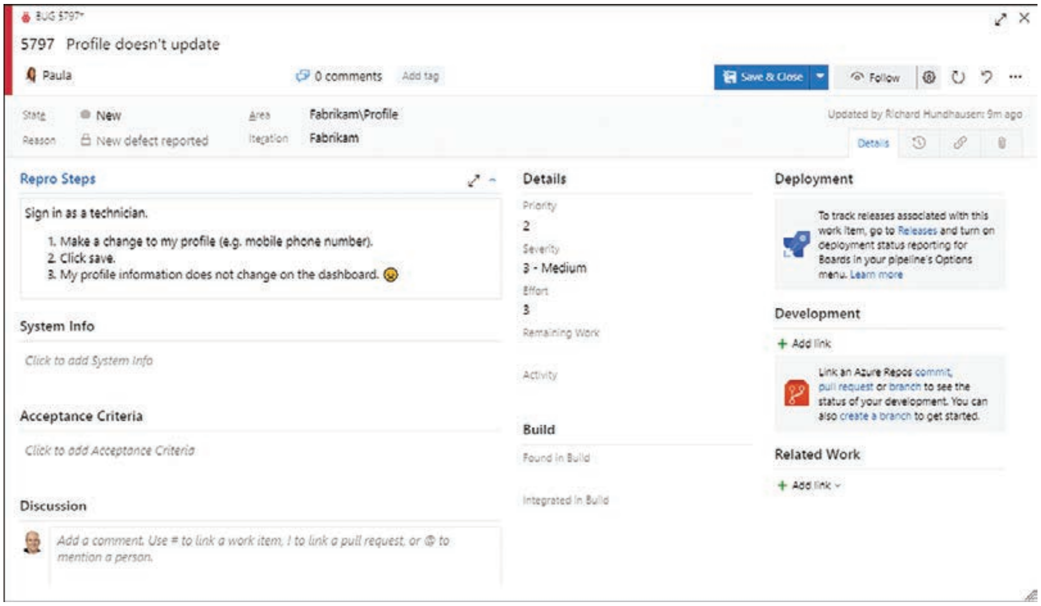
Smell It's a smell when I see tasks created and associated with a PBI prior to Sprint Planning. Perhaps the Scrum Team knows what the plan will be, but what if it changes? The creation and management of those tasks will be wasted time and, what's worse, stubborn Developers may want to stick to their archaic plan, even though conditions might have changed. To avoid this pain and waste, don't create tasks until Sprint Planning where those PBIs are forecast or later in the Sprint.

A PBI work item can be in one of five states: New, Approved, Committed, Done, or Removed. The typical workflow progression would be New ⇒ Approved ⇒ Committed ⇒ Done. When a PBI is created, it is in the New state. When the Product Owner decides that the PBI is valid, its state should change from New to Approved. When the Developers forecast to develop the PBI in the current Sprint, its state should change to Committed. Finally, when the PBI is done, according to the Definition of Done, the state should change to Done. The Removed state is used for situations where the Product Owner determines that the PBI is invalid for whatever reason, such as it is already in the Product Backlog, has already been developed, has gone stale, or is an utterly ridiculous idea. Deleting the work item is another option for these situations.

Bug

A bug communicates that a problem or potential problem exists in the product. A bug can be found in a product that has already been delivered to production, in a done Increment from a previous Sprint, or in the Increment being developed in the current Sprint. A bug is not—repeat not—a failed test. Failed tests simply indicate that the team is not yet done. This will be covered more in Chapter 7.

By defining and managing Bug work items, the Scrum Team can track these problems, as well as prioritize and plan the efforts to fix them. A bug could be as small as a typo in a data entry form or as large as a vulnerability that allows credit card data to be exposed. Figure 3-5 shows a Bug work item.



BUG 5797

5797 Profile doesn't update

Paula

0 comments Add tag

Save & Close Follow

State: New
Reason: New defect reported
Area: Fabrikam\Profile
Iteration: Fabrikam

Updated by Richard Hundhausen: 9m ago

Details

Repro Steps

Sign in as a technician.

1. Make a change to my profile (e.g. mobile phone number).
2. Click save.
3. My profile information does not change on the dashboard.

System Info

Click to add System Info

Acceptance Criteria

Click to add Acceptance Criteria

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

Details

Priority: 2
Severity: 3 - Medium
Effort: 3
Remaining Work:
Activity:

Build

Found in Build:
Integrated in Build:

Deployment

To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development

+ Add link

Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

Related Work

+ Add link

FIGURE 3-5 An example of a Bug work item.

Note In Scrum, a bug is just a *type of* Product Backlog Item, but Azure Boards defines a separate work item type to track bugs. The reason behind this is that the Bug work item type tracks additional, defect-specific information, such as severity, steps to reproduce, system information, and build numbers. Otherwise, the Bug and PBI work item types are fairly similar, with a few exceptions. Bug work items don't have a *Business Value* field, but they do have a *Remaining Work* field. The presence of the *Remaining Work* field allows the Bug work item to act like a task and be managed alongside tasks on the Taskboard. By default, the backlog includes both PBIs and bugs and the Taskboard—when used in accordance with my guidance—contains only tasks.

When you create a Bug work item, you want to accurately report the problem in a way that helps the reader understand its full impact. The steps to reproduce the bug should also be listed so that other

Developers can reproduce the behavior. There may be additional analysis (triage) required to confirm that it is an actual bug rather than a behavior that was by design. By defining and managing Bug work items, your team can track defects in the product in order to estimate and prioritize their resolution. As a general rule, bugs should be removed, not managed.

As you create or edit Bug work items, consider the following Professional Scrum guidance while entering data into the pertinent fields:

- **Title (Required)** Enter a short description that succinctly identifies the bug.
- **Assigned To** Select the Product Owner or leave blank, but don't assign to anyone else. This will reinforce the fact that the whole team owns the work on the bug.
- **Tags** Optionally add tag(s) to help find, filter, and identify the bug.
- **State** Select the appropriate state of the bug. States are covered later in this section.
- **Area** Select the best area path for the bug. Areas must be set up ahead of time and can represent functional, logical, or physical areas or features of the product. If the bug applies to all areas your team covers or you aren't sure of the specific area, then leave it set to its default value. For Nexus implementations, each team within a project can have its own corresponding areas as well as a default area.
- **Iteration** Select the Sprint in which the Developers forecast that they will fix the bug. If they have yet to forecast the bug, then leave it set to the default (root) value.
- **Repro Steps** Provide as much detail as necessary so that another team member can reproduce the bug and better understand the problem that must be fixed. If you use the Test & Feedback extension to create a Bug work item, this information is provided automatically from your test session.
- **System Info** Describe the environment in which the bug was found. If you use the Test & Feedback extension to create the Bug work item, this information is provided automatically from your test session.
- **Acceptance Criteria** Describe the conditions that will be used to verify whether the team has fixed the bug according to expectations. Acceptance criteria should be clear, concise, and testable. Consider using this field to document the expected results, as opposed to the actual results.
- **Discussion** Add or curate rich text comments relating to the bug. You can mention someone, a group, a work item, or a pull request as you add a comment. Professional Scrum Teams prefer higher-fidelity, in-person communication instead.
- **Severity** Since the Bug work item type doesn't have a *Business Value* field, you will need to instead select the value that indicates the impact that the bug has on the product or stakeholders. The range is from 1 (critical) to 4 (low). Lower values indicate a higher severity. The default severity is 3 (medium).
- **Effort** Enter a number that indicates a relative rating (size) of the amount of work that will be required to fix the bug. Larger numbers indicate more effort than smaller numbers. Fibonacci numbers (story points) work well here. T-shirt sizes don't, only because this is a numeric field. Effort can be considered the (I)nvestment in ROI.

- **Found In Build** Optionally select the build in which the defect was found.
- **Integrated In Build** Optionally, select a build that incorporates the bug fix.
- **Links** Add a link to one or more work items or resources (build artifacts, code branches, commits, pull requests, tags, GitHub commits, GitHub issues, GitHub pull requests, test artifacts, wiki pages, hyperlinks, documents, and version-controlled items). You can link the bug to a related bug, to an article explaining the root cause, to the original PBI that failed, or even to a parent PBI that serves to gather several bugs into one collective "fix" user story.
- **Attachments** Attach one or more files that provide more details about the bug. Some teams like to attach notes, whiteboard photos, or even audio/video recordings. This could also include screenshots, action recordings, and video, which the Test & Feedback extension can provide automatically.
- **History** Every time a team member updates the work item, Azure Boards tracks the team member who made the change and the fields that were changed. This tab displays a history of all those changes. The contents are read-only.

Just like a PBI, a Bug work item progresses on its journey to "ready" for Sprint Planning, the previous list of fields are really the only ones that need to be considered and completed. If there are other fields on your Bug work item form, you should discuss as a team whether or not you should be using them because tracking data in those fields is most likely waste. When the bug is forecasted to be fixed, additional fields and links will start to emerge, including links to task and test case work items, test results, commits, and builds.

A Bug work item, like the PBI work item, can be in one of five states: New, Approved, Committed, Done, or Removed. The typical workflow progression would be New ⇒ Approved ⇒ Committed ⇒ Done. When a bug is reported and determined to be genuine (that is, it's not a feature, a duplicate, or a training issue), a new Bug work item is created in the New state. When the Product Owner decides that the bug is valid, its state should change from New to Approved. When the Developers forecast to fix the bug in the current Sprint, its state should change to Committed. Finally, when the bug is done, according to the Definition of Done, the state should change to Done. The Removed state is used for situations where the Product Owner determines that the bug is invalid for whatever reason, such as it's already in the Product Backlog, it's actually a feature, it's a training issue, it's not worth the effort, or it has already been fixed. Deleting the work item is another option for these situations.

Fabrikam Fiber Case Study

Because the Bug work item type does not have a *Business Value* field and also contains several extraneous fields, Paula has decided not to use that work item type. This is not to say that the Product Backlog won't contain bugs, but rather that the Scrum Team will use the PBI work item type to track them. They will tag the PBIs accordingly, and put the repro steps and system information into the *Description* field. By doing this, the Product Backlog will contain only Product Backlog Item work items and each will have a *Business Value* and a *Size* field to compute ROI.

Epic

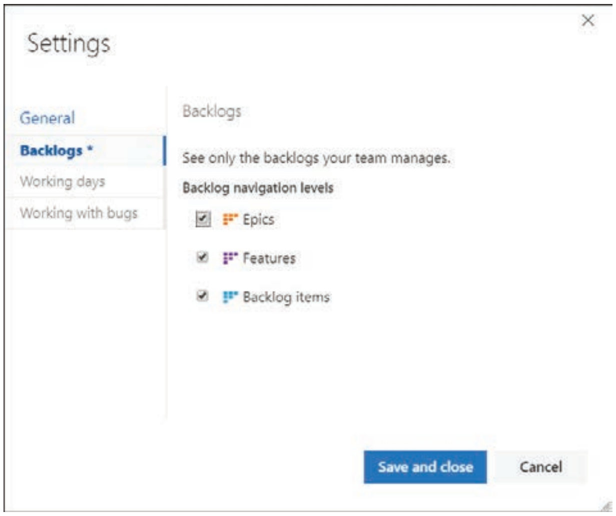
In Scrum, there is only one Product Backlog for a product and it contains only Product Backlog items. Some PBIs are quite small, deliverable in a single Sprint or less. Other PBIs are larger and may take more than one Sprint to complete. Huger PBIs may take many Sprints, even up to a year or more to complete. In Scrum, regardless of size, each item is simply called a Product Backlog item.

Organizations and teams prefer to have more specific language. They also prefer to have separate backlogs for these different-sized items, and that's why Azure Boards provides hierarchical backlogs. With hierarchical backlogs, an organization or team can start with "big picture" ideas called *epics* and break them down into more releasable-sized items called *features*, and finally into smaller, more executable-sized items.

An epic represents a business initiative to be accomplished, like these examples:

- Increase customer engagement
- Improve and simplify the user experience
- Implement microservices architecture to improve agility
- Integrate with SAP
- Native iPhone app

Note Epics and features are managed on their own backlogs. In Azure Boards, each team can determine the backlog levels that they want to use. For example, Scrum Teams may want to focus only on their Product Backlog and the higher-level Features backlog. Leadership may want to only see epics and maybe how they map to features. By default, the Epics backlog is not visible in Azure Boards. A team administrator must enable it before you can view and manage epics on that backlog, as you see in Figure 3-6.



See only the backlogs your team manages.

Backlog navigation levels

- [✓] Epics
- [✓] Features
- [✓] Backlog items

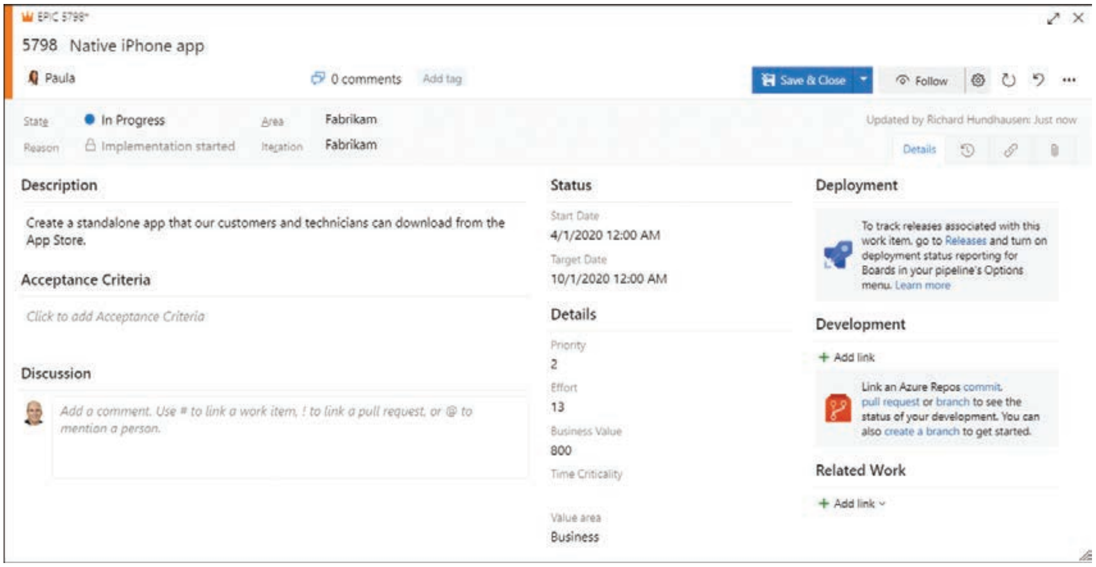
FIGURE 3-6 Enabling the Epics backlog.

Epic work items are similar to PBI work items. As you create or edit Epic work items, consider the following Professional Scrum guidance while entering data into the pertinent fields:

- **Title (Required)** Enter a short description that succinctly identifies the epic.
- **Assigned To** Select the Product Owner or leave blank. Alternatively, you can assign it to the stakeholder advocating for the epic.
- **Tags** Optionally add tag(s) to help find, filter, and identify the epic.
- **State** Select the appropriate state of the epic. States are covered later in this section.
- **Area** Select the best area path for the epic. Areas must be set up ahead of time and can represent functional, logical, or physical areas or features of the product. If the area applies to all areas your team covers or you aren't sure of the specific area, then leave it set to its default value. For Nexus implementations, each team within a project can have its own corresponding areas as well as a default area.
- **Iteration** Optional, but you can select the Sprint in which the Developers forecast that they will either begin or complete the development of the epic. If they have yet to begin work, then leave it set to the default (root) value.
- **Description** Provide as much detail as necessary so that another team member or stakeholder can understand the purpose and goal of the epic.
- **Acceptance Criteria** Describe the conditions that will be used to verify whether the team has developed the epic according to expectations.
- **Discussion** Add or curate rich text comments relating to the epic. You can mention someone, a group, a work item, or a pull request as you add a comment. Professional Scrum Teams prefer higher-fidelity, in-person communication instead.
- **Start Date** Optional, but you can set the date that work will commence on the epic. This could be the start date of the Sprint when the first PBI related to the epic is forecast for development. This field is key to using Delivery Plans.
- **Target Date** Optional, but you can set the date that the epic should be implemented. This field is key to using Delivery Plans.
- **Effort** Enter a number that indicates a relative rating (size) of the amount of work that will be required to develop the epic. Larger numbers indicate more effort than smaller numbers. Fibonacci numbers (story points) work well here. T-shirt sizes don't, only because this is a numeric field. Effort can be considered the (I)nvestment in ROI.
- **Business Value** Enter a number that indicates a fixed or relative value of delivering the epic. Larger numbers indicate more value than smaller numbers. Fibonacci numbers work well here. Business Value can be considered the (R)eturn in ROI.

- **Links** Add a link to one or more work items or resources (build artifacts, code branches, commits, pull requests, tags, GitHub commits, GitHub issues, GitHub pull requests, test artifacts, wiki pages, hyperlinks, documents, and version-controlled items). You should avoid explicitly linking epics to other epics, features, or PBIs using the Links tab. Instead, use drag and drop to establish hierarchical relationships on the backlog using the Mapping pane.
- **Attachments** Attach one or more files that provide more details about the epic. Some teams like to attach notes, whiteboard photos, or even audio/video recordings.
- **History** Every time a team member updates the work item, Azure Boards tracks the team member who made the change and the fields that were changed. This tab displays a history of all those changes. The contents are read-only.

Epic work items can be in one of four states: New, In-Progress, Done, or Removed. The typical workflow progression would be New ⇒ In-Progress ⇒ Done. When an epic is created it is in the New state. Once work begins, you should move it to the In Progress state, as I'm doing in Figure 3-7. Finally, when the epic is finished, because the last related feature is complete, then the state should change to Done. The Removed state is used for situations where the Product Owner determines that the epic is no longer needed, for whatever reason. Deleting the work item is another option in this situation.



5798 Native iPhone app

Paula 0 comments Add tag

State: In Progress Area: Fabrikam
Reason: Implementation started Iteration: Fabrikam

Updated by Richard Hundhausen: Just now

Description

Create a standalone app that our customers and technicians can download from the App Store.

Acceptance Criteria

Click to add Acceptance Criteria

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

Status

Start Date
4/1/2020 12:00 AM
Target Date
10/1/2020 12:00 AM

Details

Priority
2
Effort
13
Business Value
800
Time Criticality

Value area
Business

Deployment

To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development

+ Add link

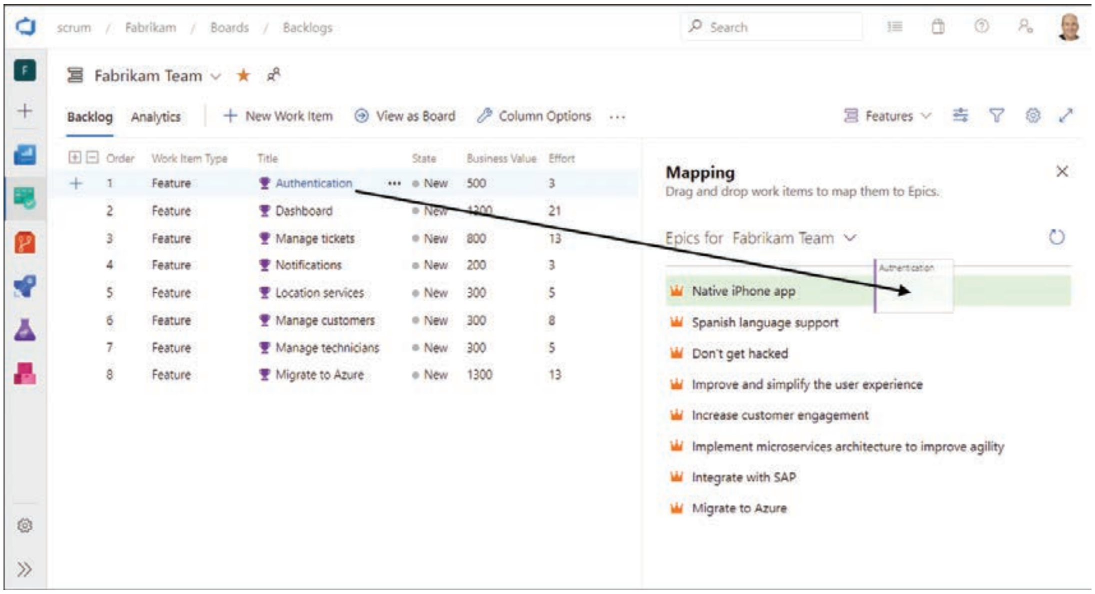
Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

Related Work

+ Add link

FIGURE 3-7 Setting an epic to In Progress.

Refining an epic means to break it down, or *decompose*, into one or more features. Feature work items are then created and linked back to the parent epic. This can be done manually by using the links in the work item form; inline on the Epics backlog; or by using the Mapping feature, as I'm doing in Figure 3-8. Refining is an ongoing process, with the features changing, merging, and splitting again as the Scrum Team learns more about the domain, the product, and the stakeholders.


scrum / Fabrikam / Boards / Backlogs

Fabrikam Team

Backlog | Analytics | New Work Item | View as Board | Column Options | Features

Order	Work Item Type	Title	State	Business Value	Effort
1	Feature	Authentication	New	500	3
2	Feature	Dashboard	New	1200	21
3	Feature	Manage tickets	New	800	13
4	Feature	Notifications	New	200	3
5	Feature	Location services	New	300	5
6	Feature	Manage customers	New	300	8
7	Feature	Manage technicians	New	300	5
8	Feature	Migrate to Azure	New	1300	13

Mapping
Drag and drop work items to map them to Epics.

Epics for Fabrikam Team

- Native iPhone app
- Spanish language support
- Don't get hacked
- Improve and simplify the user experience
- Increase customer engagement
- Implement microservices architecture to improve agility
- Integrate with SAP
- Migrate to Azure

FIGURE 3-8 Mapping a Feature work item to its parent epic.

Feature

Whether or not you plan on using Epic work items, your team may still want to track features. Features are typically what stakeholders request and also what they expect to be delivered. If a feature is larger than can be delivered in a Sprint, then it must be broken down further—into other features or into more executable-sized items that are tracked and managed at the backlog level. Most teams I work with refer to these lowest, leaf-level items as user stories, or simply stories.

A feature typically represents a releasable component of software, like these examples:

- View technician details on the dashboard
- Ability to reassign tickets
- Support text alerts
- Find and filter tickets

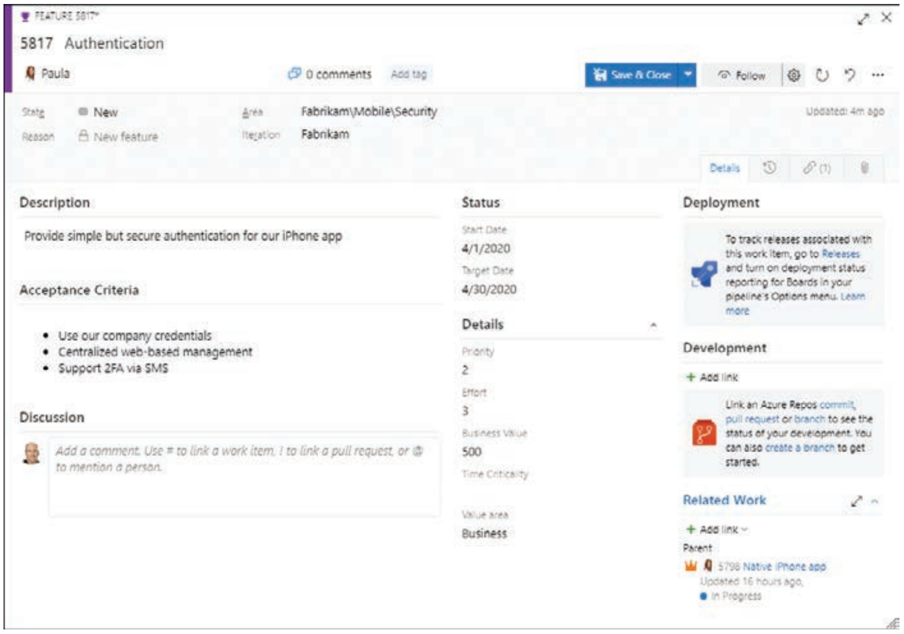
Feature work items are similar to Epic work items, as you can see in Figure 3-9. As you create or edit Feature work items, consider the following Professional Scrum guidance while entering data into the pertinent fields:

- **Title (Required)** Enter a short description that succinctly identifies the feature.
- **Assigned To** Select the Product Owner or leave blank, but don't assign to anyone else. This will reinforce the fact that the whole team owns the work on the feature.
- **Tags** Optionally add tag(s) to help find, filter, and identify the feature.

- **State** Select the appropriate state of the feature. States are covered later in this section.
- **Area** Select the best area path for the feature. Areas must be set up ahead of time and can represent functional, logical, or physical areas or features of the product. If the area applies to all areas your team covers or you aren't sure of the specific area, then leave it set to its default value. For Nexus implementations, each team within a project can have its own corresponding areas as well as a default area.
- **Iteration** Optional, but you can select the Sprint in which the Developers forecast that it will either begin or complete the development of the feature. If they have yet to begin work, then leave it set to the default (root) value.
- **Description** Provide as much detail as necessary so that another team member or stakeholder can understand the purpose and goal of the feature.
- **Acceptance Criteria** Describe the conditions that will be used to verify whether the team has developed the feature according to expectations.
- **Discussion** Add or curate rich text comments relating to the feature. You can mention someone, a group, a work item, or a pull request as you add a comment. Professional Scrum Teams prefer higher-fidelity, in-person communication instead.
- **Start Date** Optional, but you can set the date that work will commence on the feature. This could be the start date of the Sprint when the first PBI related to the feature is forecasted for development. If using epics, then the epic's start date might coincide with the first related feature's start date. This field is key to using Delivery Plans.
- **Target Date** Optional, but you can set the date that the feature should be implemented. If using epics, then the epic's target date might coincide with the last related feature's start date. This field is key to using Delivery Plans.
- **Effort** Enter a number that indicates a relative rating (size) of the amount of work that will be required to develop the feature. Larger numbers indicate more effort than smaller numbers. Fibonacci numbers (story points) work well here. T-shirt sizes don't, only because this is a numeric field. Effort can be considered the (I)nvestment in ROI.
- **Business Value** Enter a number that indicates a fixed or relative value of delivering the feature. Larger numbers indicate more value than smaller numbers. Fibonacci numbers work well here. Business Value can be considered the (R)eturn in ROI.
- **Links** Add a link to one or more work items or resources (build artifacts, code branches, commits, pull requests, tags, GitHub commits, GitHub issues, GitHub pull requests, test artifacts, wiki pages, hyperlinks, documents, and version-controlled items). You should avoid explicitly linking features to other features, epics, or PBIs using the Links tab. Instead, use drag and drop to establish hierarchical relationships on the backlog using the Mapping pane.

- **Attachments** Attach one or more files that provide more details about the feature. Some teams like to attach notes, whiteboard photos, or even audio/video recordings.
- **History** Every time a team member updates the work item, Azure Boards tracks the team member who made the change and the fields that were changed. This tab displays a history of all those changes. The contents are read-only.

Feature work items can be in one of four states: New, In-Progress, Done, or Removed. The typical workflow progression would be New ⇒ In-Progress ⇒ Done. When a feature is created it is in the New state. Once work begins, you should move it to the In-Progress state. Finally, when the feature is finished, because the last related PBI is complete, then the state should change to Done. The Removed state is used for situations where the Product Owner determines that the feature is no longer needed, for whatever reason. Deleting the work item is another option in this situation.


FEATURE 5817*

5817 Authentication

Paula | 0 comments | Add tag

State	New	Area	Fabrikam\Mobile\Security
Reason	New feature	Iteration	Fabrikam

Updated: 4m ago

Description

Provide simple but secure authentication for our iPhone app

Acceptance Criteria

- Use our company credentials
- Centralized web-based management
- Support 2FA via SMS

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

Status

Start Date
4/1/2020
Target Date
4/30/2020

Details

Priority
2
Effort
3
Business Value
500
Time Criticality

Value area
Business

Deployment

To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development

+ Add link

Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

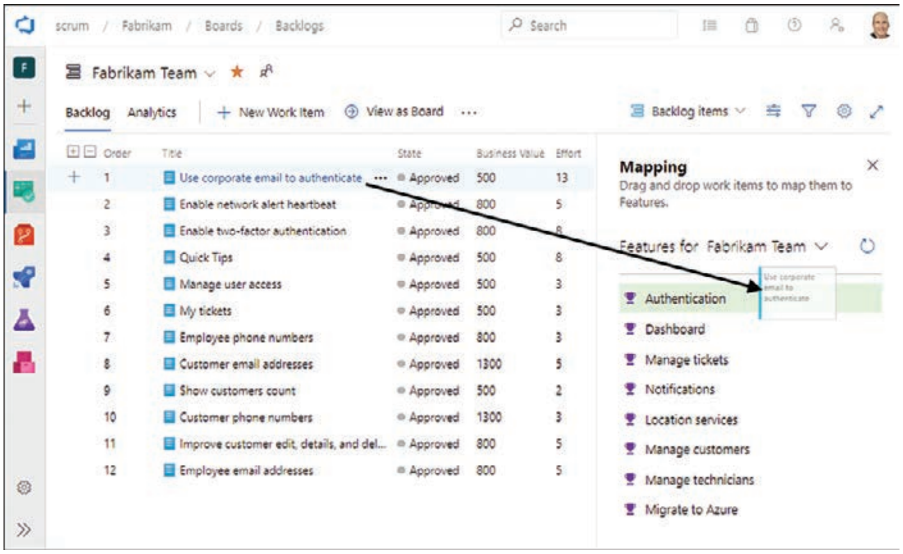
Related Work

+ Add link

Parent
5798 Native iPhone app
Updated 16 hours ago,
● In Progress

FIGURE 3-9 Creating a Feature work item (notice the related epic in the lower right).

Refining a feature means to break it down, or *decompose*, into one or more PBI work items. PBI work items are then created and linked back to the parent feature. This can be done manually by using the links in the work item form; inline on the Features backlog; or by using the Mapping feature, as I'm doing in Figure 3-10. Refining is an ongoing process, with the PBIs changing, merging, and possibly splitting again as the Scrum Team learns more about the domain, the product, and the stakeholders.


scrum / Fabrikam / Boards / Backlogs

Fabrikam Team

Backlog | Analytics | + New Work Item | View as Board | Backlog items

Order	Title	State	Business Value	Effort
1	Use corporate email to authenticate	Approved	500	13
2	Enable network alert heartbeat	Approved	800	5
3	Enable two-factor authentication	Approved	800	8
4	Quick Tips	Approved	500	8
5	Manage user access	Approved	500	3
6	My tickets	Approved	500	3
7	Employee phone numbers	Approved	800	3
8	Customer email addresses	Approved	1300	5
9	Show customers count	Approved	500	2
10	Customer phone numbers	Approved	1300	3
11	Improve customer edit, details, and del...	Approved	800	5
12	Employee email addresses	Approved	800	5

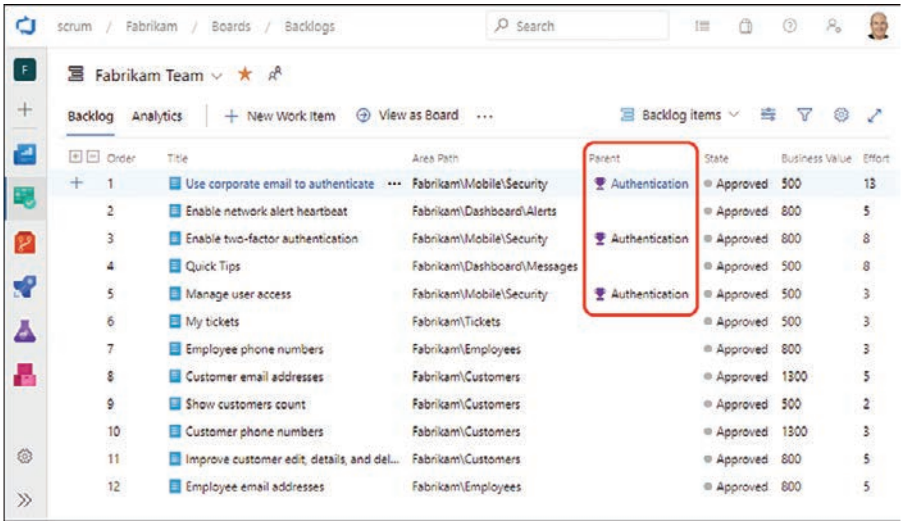
Mapping
Drag and drop work items to map them to Features.

Features for Fabrikam Team

- Authentication
- Dashboard
- Manage tickets
- Notifications
- Location services
- Manage customers
- Manage technicians
- Migrate to Azure

FIGURE 3-10 Mapping a PBI work item to its parent Feature work item.

Tip Azure Boards provides a few ways of visualizing and filtering by parent work items in the hierarchical backlogs. One option is to show parents in the backlog as nested read-only rows in the backlog. This option also includes an *unparented* section for those work items that don't have parents. These extra rows, while informational, can make your backlog view quite messy, especially if you have many parent rows. Another option is to select Column Options and add a *Parent* column, which will simply show the parent work item's title as a virtual field, as you see in Figure 3-11.


scrum / Fabrikam / Boards / Backlogs

Fabrikam Team

Backlog | Analytics | + New Work Item | View as Board | Backlog items

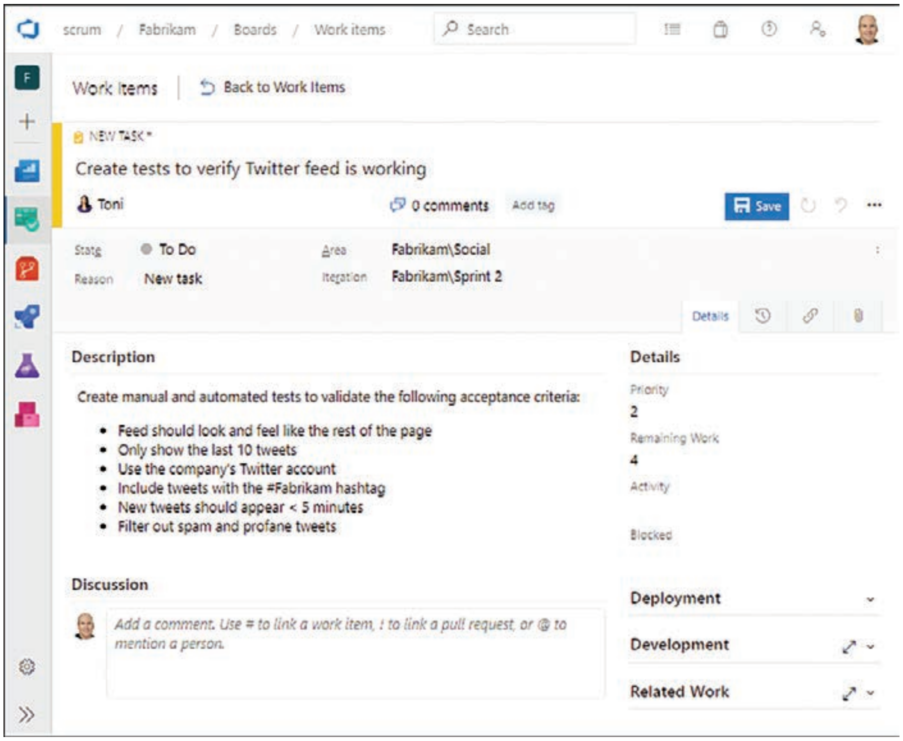
Order	Title	Area Path	Parent	State	Business Value	Effort
1	Use corporate email to authenticate	Fabrikam\Mobile\Security	Authentication	Approved	500	13
2	Enable network alert heartbeat	Fabrikam\Dashboard\Alerts		Approved	800	5
3	Enable two-factor authentication	Fabrikam\Mobile\Security	Authentication	Approved	800	8
4	Quick Tips	Fabrikam\Dashboard\Messages		Approved	500	8
5	Manage user access	Fabrikam\Mobile\Security	Authentication	Approved	500	3
6	My tickets	Fabrikam\Tickets		Approved	500	3
7	Employee phone numbers	Fabrikam\Employees		Approved	800	3
8	Customer email addresses	Fabrikam\Customers		Approved	1300	5
9	Show customers count	Fabrikam\Customers		Approved	500	2
10	Customer phone numbers	Fabrikam\Customers		Approved	1300	3
11	Improve customer edit, details, and del...	Fabrikam\Customers		Approved	800	5
12	Employee email addresses	Fabrikam\Employees		Approved	800	5

FIGURE 3-11 Displaying the parent work item title in the backlog.

Task

A Task work item represents a piece of detailed work that Developers must accomplish when developing a PBI. All tasks form the Sprint *plan* for achieving the Sprint Goal. These tasks, along with their associated PBIs, constitute the Sprint Backlog.

A task can be analysis, design, development, testing, documentation, deployment, or operations in nature. For example, the team can identify and create Task work items that are development focused, such as implementing an interface or creating a database table. They can also create testing-focused tasks, such as creating a test plan and running tests. A deployment-focused task might be to provision a set of virtual machines for hosting the deployed application. Figure 3-12 shows an example Task work item.



Create tests to verify Twitter feed is working

Toni

0 comments

State: To Do
Reason: New task
Area: Fabrikam\Social
Iteration: Fabrikam\Sprint 2

Description

Create manual and automated tests to validate the following acceptance criteria:

- Feed should look and feel like the rest of the page
- Only show the last 10 tweets
- Use the company's Twitter account
- Include tweets with the #Fabrikam hashtag
- New tweets should appear < 5 minutes
- Filter out spam and profane tweets

Details

Priority: 2
Remaining Work: 4
Activity:
Blocked:

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

FIGURE 3-12 Creating a Task work item.

As you create or edit Task work items, consider the following Professional Scrum guidance while entering data in the pertinent fields:

- **Title (Required)** Enter a short description that provides a concise overview of the task. The title should be short but descriptive enough to allow the team to quickly understand what work is to be performed. Some teams have adopted a simple verb-noun naming convention (e.g., Create tests, Write code, Deploy app, etc.).
- **Assigned To** Select the team member who is responsible for ensuring that the task is completed. A task can be assigned to only one person at a time, so if two people pair up on a task,

or the team mobs on a task, just pick one of them to be the owner. Leave it blank until someone starts working on it.

- **Tags** Optionally add tag(s) to help find and identify the task.
- **State** Select the appropriate state of the task. States are covered later in this section.
- **Area** (optional) Typically matches the associated PBI that you are working on. When tasks are created from the Taskboard, the Area is automatically populated with the parent PBI's area.
- **Iteration** Select the Sprint in which your team will be working on the task. The Sprint should be the same as the associated PBI. When tasks are created from the Taskboard, the Iteration is automatically populated.
- **Description** (optional) Provide as much detail as is necessary so that another team member can understand the nature of work to be performed in the task. A meaningful title might be sufficient. Some teams like to track task-level acceptance criteria for particularly complex tasks in this field. Avoid using this field as a repository for detailed requirements, specifications, or designs.
- **Discussion** Add or curate rich text comments relating to the task. You can mention someone, a group, a work item, or a pull request as you add a comment. Professional Scrum Teams prefer higher-fidelity, in-person communication instead.
- **Remaining Work** The estimated hours of work remaining to complete the task.

Tip Initially, during Sprint Planning, the Remaining Work value should be an estimated provided by the entire team. Later, after a team member begins working on the task, it should be updated by that person, who has more up-to-date knowledge of the work. Ideally, tasks should be 8 hours or less. If a task is going to take longer than 8 hours, it should be decomposed into smaller tasks, in order to reduce risk and enable more collaboration options. Remaining work estimates should be updated daily.

- **Blocked** (optional) Indicates whether the task is blocked from being accomplished. Blocked work should be identified and mitigated immediately. Instead of using the Blocked field, some teams have opted to use a "Blocked" tag.
- **Links** Add a link to one or more work items or resources (build artifacts, code branches, commits, pull requests, tags, GitHub commits, GitHub issues, GitHub pull requests, test artifacts, wiki pages, hyperlinks, documents, and version-controlled items). In general, you should avoid manually linking tasks to other PBIs, preferring to use the Sprint Backlog or Taskboard instead. Linking tasks to other tasks can help visualize dependencies, but it also has the smell of a command-and-control work breakdown structure.
- **Attachments** Attach one or more files that provide more details about the task. Some teams like to attach notes, whiteboard photos, or even audio/video recordings.

- **History** Every time a team member updates the work item, Azure Boards tracks the team member who made the change and the fields that were changed. This tab displays a history of all those changes. The contents are read-only.

As your team uses tasks to plan, visualize, and manage its Sprint work, the previous list contains the only fields that you need to consider and complete. If there are other fields on your Task work item form, such as Priority or Activity, you should discuss as a team whether or not you should be using them because tracking data in those fields is most likely waste. That said, at the end of the day, *how* the team works, which includes how they will use Azure Boards, is up to them—which is an example of self-management.

Smell It's a smell when I see that a team is using the *Activity* field on tasks. Professional Scrum Teams know that everything they do is considered a *development* activity, so using this field seems like waste. There is also a risk that Developers will become conditioned to look for their favorite type of task. For example, someone with a background in testing may only look for unassigned testing tasks, which is not necessarily what is best for the team's productivity, let alone achieving the Sprint Goal. An even greater fear is that others outside the Scrum Team will begin using the activity type for resource planning or assignment of work!

A Task work item can be in one of four states: To Do, In Progress, Done, or Removed. The typical workflow progression would be To Do ⇒ In Progress ⇒ Done. When a task is created, it is in the To Do state. When a team member begins working on a task, the state should be set to In Progress. When the task is finished, the state should be set to Done. The Removed state is used for situations where the Developers determine that the task is invalid for whatever reason, such as it doesn't apply anymore or it was a duplicate. Deleting the work item is another option for these situations.

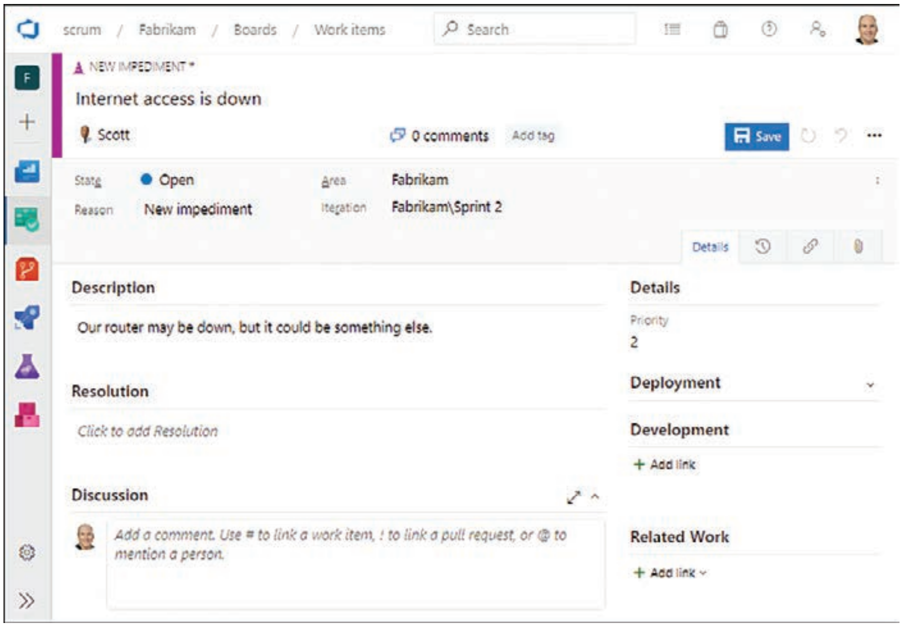
Impediment

An Impediment work item is a report of any situation that blocks the team or a team member from completing work efficiently. By defining and managing Impediment work items, a Scrum Team can identify and track problems that are blocking it. More importantly, they'll have a backlog from which to work on improvements.

Impediments can be identified and, optionally recorded, at any time. They should be made transparent at least once a day, perhaps during the Daily Scrum. Professional Scrum Teams, however, don't wait until the Daily Scrum to raise and/or fix impediments. If the impediment is something that can be removed immediately, that's what should be done. If not, then the impediment could be recorded as an Impediment work item. The Scrum Team may also record impediments on a physical board or on a wiki page. Regardless, it's better to remove impediments than to track and manage them. The Scrum Master is responsible for facilitating the resolution of impediments—that the team cannot resolve themselves—as well as improving team productivity.

Tip Having a transparent, prioritized backlog of impediments and improvement ideas at a team's fingertips can be very beneficial. If and when management comes offering to help, an item can be pulled from the top and discussed. Even if the budget is tight and management can't afford new hardware, software, or services, you may still have impediments that don't require an expenditure. For example, management may not have any money for faster laptops, but they could ask the Project Management Office (PMO) to ease off on the weekly status report requirements.

Figure 3-13 shows you an example of an Impediment work item.



Internet access is down

Scott

0 comments

State: Open
Reason: New impediment
Area: Fabrikam
Iteration: Fabrikam\Sprint 2

Description

Our router may be down, but it could be something else.

Resolution

Click to add Resolution

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

Details

Priority
2

Deployment

Development

+ Add link

Related Work

+ Add link

FIGURE 3-13 An example of an Impediment work item.

As you create or edit Impediment work items, consider the following Professional Scrum guidance while entering data in the pertinent fields:

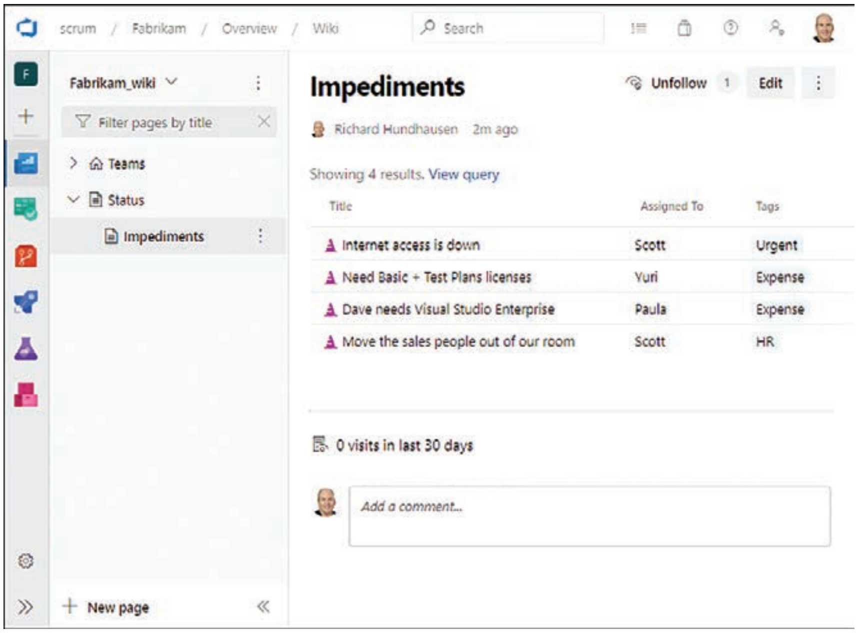
- **Title (Required)** Enter a short description that accurately and succinctly describes the impediment.
- **Assigned To** Select the team member or stakeholder who will be responsible for resolving the impediment. Don't assume that the Scrum Master will always own/remove impediments.
- **Tags** Optionally add tag(s) to help find and identify the impediment.
- **State** Select the appropriate state of the impediment. States are covered later in this section.
- **Area** (optional) Select the best area for the impediment. If the impediment applies to all areas your team covers or you aren't sure of the specific area, then leave it set to its default value.

- **Iteration** (optional) Typically a team selects the Sprint in which the impediment occurred, but Iteration could also represent the Sprint in which the impediment was removed. Leaving it set to its default value is fine as well.
- **Description** Provide as much detail as necessary so that another person can understand the impediment and its impact.
- **Resolution** Provide as much detail as necessary to describe how the impediment was resolved. Over time, these resolutions could establish a "lessons learned" reference.
- **Discussion** Add or curate rich text comments relating to the impediment. You can mention someone, a group, a work item, or a pull request as you add a comment. Professional Scrum Teams prefer higher-fidelity, in-person communication instead.
- **Priority** Select the level of importance for the impediment on a scale of 1 (most important) to 4 (least important). The default value is 2.
- **Links** Add a link to one or more work items or resources (build artifacts, code branches, commits, pull requests, tags, GitHub commits, GitHub issues, GitHub pull requests, test artifacts, wiki pages, hyperlinks, documents, and version-controlled items). For example, you may want to link the impediment to one or more blocked tasks or PBIs or other impediments.
- **Attachments** Attach one or more files that provide more details about the impediment. Some teams like to attach notes, whiteboard photos, or even audio/video recordings.
- **History** Every time a team member updates the work item, Azure Boards tracks the team member who made the change and the fields that were changed. This tab displays a history of all those changes. The contents are read-only.

Note Impediments may seem similar to tasks, and vice versa. To add further confusion, impediments are referred to as *issues* in other processes, and issues in the Basic process represent work to be done. To keep it straight in your head, consider this simple definition of an impediment—which is anything that hinders or prevents you or your team from achieving the Sprint Goal. In other words, Impediment work items are used to track unplanned situations that *block* work from getting done, whereas Task work items represent the plan for developing the forecasted PBIs in the Sprint Backlog and achieving the Sprint Goal.

An Impediment work item can be either Open or Closed. When an impediment is created, it is in the Open state. When the impediment is resolved/removed, the state should be set to Closed. Deleting the impediment after it has been removed is another option.

Impediments can be configured to show on the Boards. You can also track and manage them using a work item query. A team administrator could create a shared query looking for Work Item Type of Impediment and State of Open sorted by Priority. This query could then be surfaced on a dashboard or a wiki page, as I've done in Figure 3-14.



Impediments

Richard Hundhausen 2m ago

Showing 4 results. View query

Title	Assigned To	Tags
Internet access is down	Scott	Urgent
Need Basic + Test Plans licenses	Yuri	Expense
Dave needs Visual Studio Enterprise	Paula	Expense
Move the sales people out of our room	Scott	HR

0 visits in last 30 days

Add a comment...

FIGURE 3-14 Displaying open impediments on a wiki page.

Scrum Work Item Queries

Work item queries allow you to view, understand, and manage your workload. By running the appropriate query, you can return lists of work items showing you the PBIs, bugs, tasks, impediments, test cases, and other work items that pertain to you or your team. You can filter and sort those items in many ways. You can then decide on which of these work items to take action. Queries can also be used to perform bulk work item updates. For example, the Product Owner can query those PBIs in a specific area in order to make bulk changes to the *Business Value* field.

With queries, you can perform these functions:

- Review work that's planned, in progress, or recently done
- Perform bulk updates, such as assigning new PBIs to the Product Owner
- Create a chart to get a count of items or the sum of a field
- Create a chart and add it to a dashboard
- View a tree of parent-child-related work items

Note Queries can be executed from Microsoft Excel and other clients. When you have many work items to add or modify, Excel can really save you time. Simply create a flat list query of epics, features, PBIs, bugs, or tasks and open it in Excel. You must first install the (free) Azure DevOps Office Integration plug-in, which supports Microsoft Excel 2010 or later versions, including Microsoft Office Excel 365.

When saving a query, you can save it to *My Queries* or, if you have permissions, save it to *Shared Queries*. As you might guess, only you can view and run queries saved under *My Queries*. Queries that you and others save under *Shared Queries* can be viewed by everyone with access to the project. Queries can be organized within folders and even marked as favorites.

Here are some queries that your Professional Scrum Team may want to create:

- **Open impediments** The Scrum Team and especially the Scrum Master should be mindful of these.
- **PBIs assigned to someone other than the Product Owner** In Scrum, the Product Owner, not anyone else, "owns" the Product Backlog items. Empty assigned-to values are okay.
- **New or approved PBIs with tasks** It is wasteful to create tasks ahead of Sprint Planning.
- **Approved PBIs without acceptance criteria** How will the team members know what the expectations are or when development is done?
- **New or approved PBIs with a root-level area** Are these items really cross-cutting, or did someone just forget to assign an area?
- **New or approved PBIs assigned to a Sprint** Either someone entered the wrong iteration or someone forgot to set the state.
- **Committed or done PBIs without a Sprint** Either someone forgot to set the iteration or someone goofed up the state.
- **PBIs without links to features** Assuming you are using features, it may be useful to see the unparented items.
- **Features without links to epics** Assuming you are using epics and features, it may be useful to see the unparented items.
- **Features without links to PBIs** Assuming you are using features, it may be useful to see the items without children.
- **Epics without links to features** Assuming you are using epics and features, it may be useful to see the items without children.

Here are some additional query ideas that pertain to the current Sprint:

- **Committed PBIs without tasks** Perhaps the plan for delivering these items really is that simple. It's more likely that the team forgot to create a plan or a PBI was snuck into the Sprint after Sprint Planning.
- **Committed PBIs with associated tasks from other Sprints** These PBIs were either rolled over from a previous Sprint with a part of the plan remaining in that Sprint, or there are some serious problems with your iteration values or planning practices.
- **Committed or done PBIs with no business value** How will the Product Owner explain the investment in something with no value? More likely, someone forgot to enter the business value.
- **Committed or done PBIs with no effort** Well, that was easy. Somebody probably forgot to enter the effort.

- **Committed or done PBIs without acceptance criteria** How will the team members know what the expectations are or when development is done?
- **To-do or in-progress tasks outside of current Sprint** Looks like a previous Sprint plan was not cleaned up correctly.
- **To-do tasks are assigned to a team member** It's better to leave to-do tasks unassigned so that any team member with capacity can help out in order to increase the chances of meeting the Sprint Goal.
- **To-do or in-progress tasks without remaining work** Assuming your team has a working agreement to estimate hours for tasks, this query can show those tasks that were overlooked.
- **To-do or in-progress tasks with remaining work > 8** Assuming your team has a working agreement that no task should take longer than 8 hours, this query can show those tasks that need to be decomposed.
- **Tasks not linked to a PBI** Not all work in the Sprint Backlog needs to pertain to developing the forecasted PBIs, but it can be a smell if there are "free-floating" tasks in there.
- **Tasks with activity set** Assuming you follow my advice in this chapter and don't see any value in using this field, then it can be helpful to see which tasks may have this accidentally set. You can then turn this into a learning opportunity.
- **Blocked tasks** Whether the Scrum Team uses the *Blocked* field or sets a tag or both, it can be useful to know which tasks are currently blocked.
- **In-progress tasks not assigned to a team member** Who's working on these tasks?
- **One team member has multiple in-progress tasks** Don't you know that multitasking is a myth and attempting it will damage your brain? Perhaps one of the tasks is actually done or blocked.
- **Team members without tasks** With the exception of the Product Owner and Scrum Master (unless they are also a Developer), everyone should be working out of the Sprint Backlog. Be careful with this query—it could become a weapon in the wrong hands.
- **Done tasks have remaining work > 0** How can you be done with a task if there is still remaining work? More than likely, this was just an oversight.
- **Done PBI has new or in-progress tasks** How can you be done with a PBI if one or more associated tasks are not done?

If the Scrum Team is using Azure Test Plans, there are a few more query ideas to consider, specifically related to testing:

- **New or approved PBIs with test cases** It is wasteful to create test cases ahead of Sprint Planning.
- **Committed PBIs without test cases** Perhaps you are proving acceptance in some other way, such as exploratory testing.

- **No test plan for current Sprint** Maybe this Sprint is exceptional and doesn't require any acceptance testing, or more likely, someone hasn't created a test plan for it yet.
- **Test cases not linked to a PBI** It's a smell to see test cases in a test plan that are not explicitly linked to one or more PBIs. Perhaps it is a cross-cutting acceptance test, but it could also be an oversight.

Scrum Guide Drift

When the Scrum process (formerly known as the Visual Studio Scrum process template) was introduced at Microsoft's TechEd North America conference in New Orleans in 2010, it *exactly* matched the *Scrum Guide*. Over the years, however, the two have drifted apart. The *Scrum Guide* evolved while the Scrum process template did not. For example, in late 2014 Microsoft went crazy and added support for the Scaled Agile Framework (SAFe) to *all* of their process templates—even our beloved Scrum one. Although this was good in that users now had additional hierarchy support in the backlog, it also added extraneous fields.

Also, in 2014 the *Scrum Guide* was moved off Scrum.org and posted to the neutral ScrumGuides.org. At the same time, all the major Scrum organizations in the world acknowledged this as the official definition of Scrum. Unfortunately, Microsoft didn't get the memo. Sure, they still have a Scrum process, but it no longer matched the *Scrum Guide* and it was no longer "barely sufficient."

Note For more than a decade, survey after survey has demonstrated that agile is and remains the most popular and successful way to develop software. Those same surveys also show that Scrum is the most popular framework to become agile—always in the 80–90 percent range of agile organizations. With this in mind, Microsoft should make the Scrum process the *default* process when creating a project. It used to be.

Over the years the Professional Scrum community has maintained in close relationship with Microsoft, and we've done what we could to keep the Scrum process from drifting too far from the *Scrum Guide*. In this section, I will explore the current differences between the two.

Work Item Types

Azure DevOps offers more than a dozen work item types—most of which don't particularly relate to planning and managing work. Therefore, I will focus only on those work item types that I have previously listed in the Scrum process section.

- **Bug** The *Scrum Guide* does not mention bugs at all. That's because a bug is a *type of* PBI. The confusing part is that the Scrum process also includes a PBI work item type. In my opinion, the only reason the Bug work item type exists is so that tooling such as the Test & Feedback extension can create a specific work item with repro steps and system information—both of which could be tracked in a PBI work item's description field.

- **Epic** and **Feature** Again, the *Scrum Guide* only mentions Product Backlog items. It doesn't mention epics and it doesn't mention features. Microsoft did this back in 2014 to support SAFe. Professional Scrum Teams using Azure Boards have since become comfortable with hierarchical backlogs, even though they could have engineered these backlogs to use the existing PBI work item type on all backlog levels.

Backlog Levels

As I have mentioned, Microsoft introduced hierarchical backlogs to support scaled agile practices. If they had kept the PBI work item type at each backlog level, that would have kept in alignment with Scrum. But since they didn't, we now have epics, features, and PBI work item types, and the result is a goofy mix of terminology.

If organizations and teams want to use the hierarchical backlogs, and most of the ones I consult with do, then perhaps Microsoft could rename the lowest leaf-level "Backlog items" to something like "Stories"—which is the most popular term I see used. In this way, it's made clearer that the names of the backlog levels are all *not* Scrum but more industry-standard names for types of PBIs. You and your organization can refer to the items in this lowest level however you'd like.

PBI Work Item Fields

Over the years, Microsoft has added many fields to the "barely sufficient" PBI work item type. In this section, I will take a look at those fields in the PBI work item type and give my Professional Scrum opinions, including why the use of some may be considered waste.

- **Assigned To** Sounds very command-and-control. The label and underlying field should be changed to something that sounds more like a tool for self-managing teams, such as *Owned By*. Also, Azure DevOps should let you tag which user in the project is the "Product Owner" and make that the default value of this field.
- **Reason** For the Scrum process, the reasons are all read-only and weak. This field should just be removed or hidden from the form.
- **Iteration** Should always default to the root level when adding a PBI. Rarely would a Scrum Team add a PBI directly to an existing Sprint. Microsoft got carried away with the use of a team's default iteration in this regard.
- **Priority** Product Owners don't necessarily need a field for priority, as the position of the PBI in the ordered Product Backlog suggests its "priority." If a Product Owner wants to track an individual PBI's priority, it would be better to express business priority using the *Business Value* field so that all PBIs can be compared relative to each other on a common field and using a common scale like Fibonacci.
- **Effort** Invokes thoughts of specifying hours and classic project management, instead of something more abstract and better suited for complex work like Fibonacci numbers or story points. *Size* would be a better label and underlying field name.

- **Value area** Product Backlog items can have value for a number of reasons, well beyond the two options in this drop-down. What's more, teams may think that *architecture* work has value, which is rarely the case. Architectural work is required to deliver the kind of value that a stakeholder is looking for, but it's rarely of direct value itself. It's better to not use this field so that all items can have a measure of value relative across the same stratification.
- **Business Value** As a corollary to what I mentioned earlier, I think *Value* would be a much simpler and better label and underlying field name.

PBI Work Item Workflow States

One of the most controversial updates to the 2011 *Scrum Guide* was the removal of the term "commit" in favor of "forecast" in regard to the work selected for a Sprint. Prior to this change, practitioners used to say that the Development Team *commits* to the Product Backlog Items that it will deliver by the end of the Sprint. Scrum now calls that selection and practice a *forecast*—because it better reflects the reality of doing complex work in a complex domain.

Well, as you can guess, Microsoft never updated the Scrum process. The Scrum community has had to put up with the Committed workflow state, as you can see in Figure 3-15. I would welcome the change to *Forecasted*, or even *Planned*.

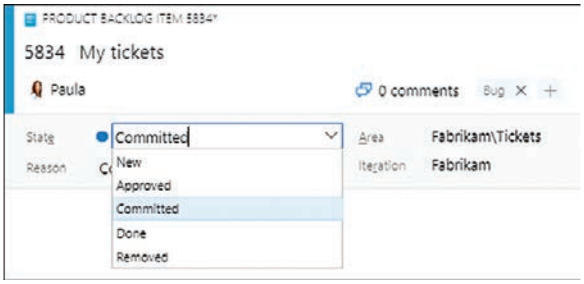


FIGURE 3-15 The Committed workflow state, another example of misalignment with the *Scrum Guide*.

Another small nit I have with the workflow states is the state of Approved. It's not bad, but *Ready* would be preferred. Although "Ready" is not an official thing in Scrum, it is mentioned in older *Scrum Guides*: "Product Backlog items that can be 'Done' by the Development Team within one Sprint are deemed 'Ready' for selection in a Sprint Planning."

Process Customization

As you've learned, each Azure DevOps project is based on a process that defines the building blocks for tracking work. Of the out-of-the-box *system* processes, the Scrum process most closely matches the *Scrum Guide*, but not entirely. There has been some drift over the past 10 years.

Fortunately, you can customize the Scrum process to make it more closely match the *Scrum Guide*, and even your own organization or team's specific needs. Achieving this requires you to create an *inherited* process first and then make customizations to that. Any changes you make to the inherited process automatically appear in the projects that use that process. You cannot make changes to the system processes.

You primarily customize a process by adding or modifying its work item types. This is done through an administrative user interface in the web portal.

The general sequence for process customization looks like this:

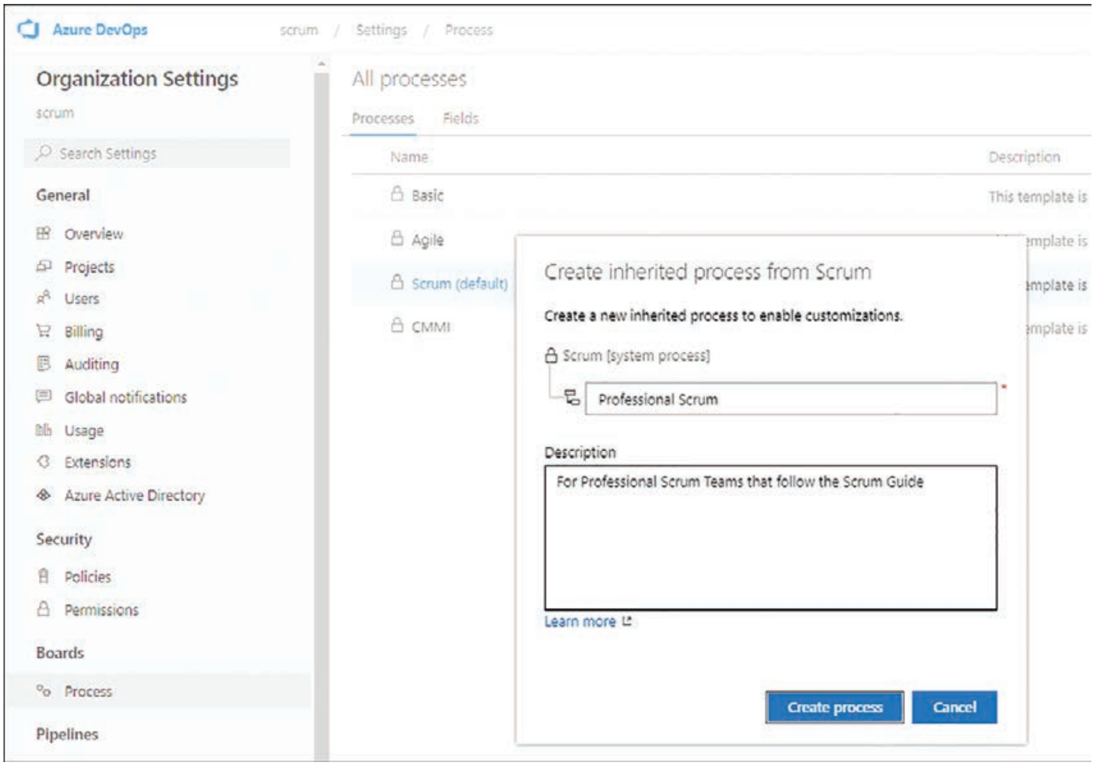
- **Create an inherited process** Select a system process (for example, Scrum) and create an inherited process (such as Professional Scrum) based on it.
- **Customize the inherited process** Add or modify work item types, work item fields, work item workflow states, and work item form UIs. You can also update backlog behavior.
- **Apply inherited process to project(s)** Create new projects using the inherited process or change existing projects to use the new inherited process.
- **Refresh and verify** Refresh the web portal and explore the changes to the work items and backlogs.

Note This section covers the inheritance process model, which is available in Azure DevOps Services and Azure DevOps Server. Legacy Team Foundation Server instances used an XML process model, which provided support for customizing work tracking objects and agile tools for a project. With this older model, you had to update the XML definitions of work item types, process configuration, categories, and more. On-premises XML process configuration is beyond the scope of this book.

Professional Scrum Process

If your organization or team cares about the *Scrum Guide* and wants to address the drift between it and the system Scrum process, you should consider following the instructions in this section to create a custom, inherited Professional Scrum process. Doing so is completely optional but may result in a better experience for Scrum teams. It will also help organizations and teams that are just adopting Scrum where precise language and terms are important for establishing a new mental model.

Process customization takes place at the organization level (or at the collection level for the on-premises Azure DevOps Server). You can select any of the system processes and create an inherited process, as I've done with the Scrum process in Figure 3-16.



Create a new inherited process to enable customizations.

Scrum [system process]

Professional Scrum

Description

For Professional Scrum Teams that follow the Scrum Guide

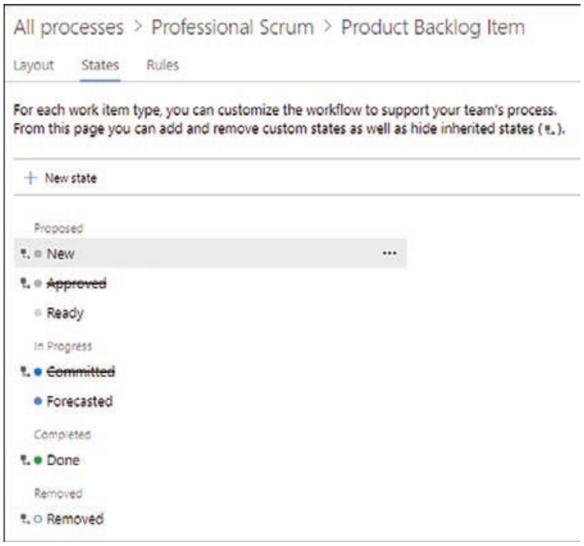
FIGURE 3-16 Creating a Professional Scrum process from the Scrum system process.

After I create the inherited Professional Scrum process and set it to be the default process, I then disable the Bug work item type. This allows the Scrum Team to use the PBI work item type for all work in the Product Backlog. Teams can add a “Bug” tag to those PBIs if they wish.

Next, I update the Product Backlog Item work item type, making the following changes:

- **Hide the *Priority* and *Value Type* fields from the layout** I would like to remove these fields altogether, but that customization isn’t allowed in Azure Boards.
- **Change the Effort label to Size** I could also create a new *Size* field behind the scenes, but I will leave the Effort field in use.
- **Change the Business Value label to “Value”** I could also create a new value field behind the scenes, but I will leave the *Business Value* field in use.
- **Rename the Details group to “ROI”** The only two fields in this group now are related to ROI. It would be awesome to include a computed ROI field, but that functionality is not available outside of using an extension.

Next, I make changes to the workflow states by adding two new states: Ready (which maps to the Proposed category) and Forecasted (which maps to the In Progress category). I keep the default colors for these new states. Next, I hide the Approved and Committed states, replacing them with the Ready and Forecasted states that I just created, as you can see in Figure 3-17.


All processes > Professional Scrum > Product Backlog Item

Layout States Rules

For each work item type, you can customize the workflow to support your team's process.
From this page you can add and remove custom states as well as hide inherited states.

+ New state

Proposed
- New
- ~~Approved~~
- Ready

In Progress
- ~~Committed~~
- Forecasted

Completed
- Done

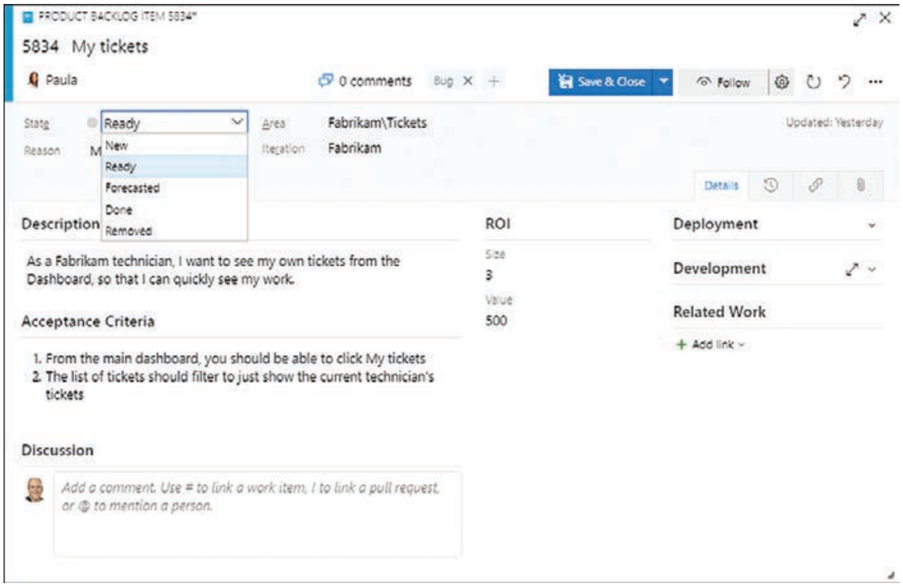
Removed
- Removed

FIGURE 3-17 Customizing PBI workflow states in the Professional Scrum process.

For organizations and teams that use the Epic and Feature work item types, you can make similar customizations by hiding those extraneous fields that you don't use (e.g., *Priority*, *Time Criticality*, *Value Area*, and even *Start Date* and *Target Date*). You could also rename the labels and normalize the workflow states, as I did for the PBI work item type.

I also hide the *Priority* and *Activity* fields from the Task work item type. The last customization I do is to rename the lowest leaf-level backlog from "Backlog items" to "Stories" (or whatever the organization/team would like it to be called). Leaving it named *Backlog items* is confusing, because in actuality, all backlog levels contain "backlog items."

After these changes are made, I can start creating projects based on the Professional Scrum process. If I have any existing projects, I can also change them to use the new Professional Scrum process, which you can see in Figure 3-18. Later, if I make any changes to the Professional Scrum process, all projects based on it will instantly reflect that change.


PRODUCT BACKLOG ITEM 5834*

5834 My tickets

Paula

0 comments

Bug

Save & Close

Follow

State	Ready
Reason	M
Area	Fabrikam\Tickets
Iteration	Fabrikam

New
Ready
Forecasted
Done
Removed

Updated: Yesterday

Details

Description

As a Fabrikam technician, I want to see my own tickets from the Dashboard, so that I can quickly see my work.

Acceptance Criteria

1. From the main dashboard, you should be able to click My tickets
2. The list of tickets should filter to just show the current technician's tickets

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

ROI

Size	3
Value	500

Deployment

Development

Related Work

+ Add link

FIGURE 3-18 PBI work item form after applying the Professional Scrum process.

Other Customizations

Beyond matching the Scrum process to the *Scrum Guide*, organizations and teams may want to make additional customizations to work items and backlogs. Here are some examples I've collected over the years from various teams and other consultants:

- Add a *Team* field to the PBI work item type to indicate which team owns it, rather than using the *Area* field gimmick.
- Add a *Value Stream* work item type and corresponding top-level backlog that sits above Epic.
- Add a *Planned Sprint* field to the Impediment work item type for planning when the improvement will be done. The system *Iteration* field can be used for the Sprint where the impediment was discovered.
- Add a new workflow state to the Impediment work item type to indicate which improvement(s) are currently *in-progress*.
- Add an *Improvement* work item type to plan and track any improvement experiments being performed.
- Add a *Remaining Work* field to the PBI work item type to store the rolled-up sum of any child task *Remaining Work* values. External automation, such as an extension, would be required to do the rolling up.
- Add default user story description text ("As a <*type of user*>, I want <*some goal*> so that <*some reason*>") to the *Description* field of the PBI work item type.

- Add default text to the PBI work item type *Acceptance Criteria* fields to suggest a "given-when-then" behavior-driven development (BDD) or "given-when-then-fail" format.
- Add a *Hypothesis* work item type to support hypothesis-driven development.

Tip Use your Scrum or Professional Scrum process the way that it was designed for a few Sprints before customizing anything. I've seen teams want to immediately make their new project look and behave like their old project or culture. For example, if after reading this book you decide to abandon your Agile process project, creating a new Professional Scrum project, don't immediately add the fields that used to be in your old project (for example, the *Original Estimate* and *Completed (hours)* fields in the Task work item type—which we removed from the Scrum process for a reason; tracking original estimates and actual hours are generally considered waste). Just know what you are doing and why you are doing it before making any "improvements" and weigh the benefits against potential waste and misuse. Don't inadvertently change the rules of Scrum by customizing the tool!

Fabrikam Fiber Case Study

The Scrum Team has decided to follow the guidance in this section and create and use the Professional Scrum process, inherited from the Scrum process. They will apply this new process to their existing Fabrikam project. This will be the process that is referenced in the coming chapters. For that reason, you may want to take a moment and create a Professional Scrum process yourself so that you can better follow along.

Chapter Retrospective

Here are the key concepts I covered in this chapter:

- **Process** When creating a project, you will need to select a process. Microsoft provides several out-of-the-box processes, referred to as system processes.
- **Scrum process** A Scrum-centric process created through a collaboration of Microsoft and the Professional Scrum community.
- **Work item types** Although there are more than a dozen Azure DevOps work item types, including a number of hidden ones, the ones that apply to planning and managing work are Product Backlog Item, Bug, Epic, Feature, Task, and Impediment. Task and Test Case work items should be created and linked only during the Sprint in which you are working on their parent PBIs.

- **Queries** There are a number of queries that a Scrum Team could create and share to track and manage the work in the Scrum development effort.
- **Scrum Guide drift** Over the years, the *Scrum Guide* has evolved while the Scrum process has not. This issue can be overcome by creating and customizing an inherited process.
- **Inherited process** A child of one of the system processes that can be customized in a structured way, inherited processes can be used to create new projects as well as applied to existing projects. Future changes to the inherited process are instantly visible in all the projects that use that process.

Index