Microsoft

Professional Scrum Development with Azure DevOps

Professional

Richard Hundhausen

Foreword by Ken Schwaber, Co-creator of Scrum

Contents

Foreword . . . . . . *xviii*

Introduction . . . . . . *xx*

PART I SCRUMDAMENTALS

Chapter 1 Professional Scrum 3

The Scrum Guide 4

The Pillars of Scrum 5

Scrum in Action 5

Scrum Roles 8

Scrum Events 16

Scrum Artifacts 30

Definition of Done 40

The Scrum Values 42

Professional Scrum 44

The Professional Scrum Developer 44

Chapter Retrospective 46

Chapter 2 Azure DevOps 49

A Brief History 49

Delivering Continuous Value 51

Azure DevOps Services 54

Azure Boards 56

Azure Repos 57

Azure Pipelines 58

Azure Test Plans 59

Azure Artifacts 60

Azure DevOps Server 61

Migrating to Azure DevOps Services 62

Visual Studio 63
Visual Studio Subscriptions 65
Azure DevOps Access Levels 66
Stakeholder Access 67
GitHub and the Future 67
Chapter Retrospective 68

Chapter 3 Azure Boards 71

Choosing a Process 71
Work Item Types 72
The Scrum Process 75
Scrum Work Item Types 75
Scrum Work Item Queries 94
Scrum Guide Drift 97
Process Customization 99
Professional Scrum Process 100
Other Customizations 103
Chapter Retrospective 104

PART II PRACTICING PROFESSIONAL SCRUM

Chapter 4 The Pre-game 109

Setting Up the Development Environment 110
Creating an Azure DevOps Organization 111
Providing Access to the Organization 112
Other Organization Configurations 114
Azure DevOps Marketplace Extensions 115
Setting Up Product Development 119
Creating a Project 119
Adding Project Members 123
Other Project Configurations 128
Establishing Information Radiators 130
Pre-game Checklist 135
Chapter Retrospective 137

Chapter 5 **The Product Backlog** 139

Creating the Product Backlog 139

Creating a Product Backlog in Azure Boards 140

Adding Product Backlog Items 143

Importing Product Backlog Items 150

Removing a Product Backlog Item 154

Effective Product Backlog Creation 155

Reporting a Bug 155

What Makes a Good Bug Report? 157

Where Do Bugs Come From? 159

In-Sprint vs. Out-of-Sprint Bugs 160

Bug Reactivations 163

Refining the Product Backlog 164

Specifying Acceptance Criteria 166

Sizing Product Backlog Items 168

Splitting Product Backlog Items 173

Definition of Ready 174

Ordering the Product Backlog 178

Planning a Release 179

Story Mapping 182

SpecMap 183

Product Backlog Checklist 185

Chapter Retrospective 186

Chapter 6 **The Sprint** 189

Sprint Planning 190

Sprinting in Azure Boards 192

Creating the Sprint Backlog 192

Creating the Forecast 193

Capturing the Sprint Goal 201

Creating the Plan 203

Sprint Activities 206

The Daily Scrum 207

Decomposing Tasks 209

The Taskboard 211
Closing Out a Sprint 227
Sprint Planning Checklist 229
Chapter Retrospective 230

Chapter 7 Planning with Tests 233

Azure Test Plans 234
Organizing Tests 234
Test Cases 237
Inspecting Progress 241
Acceptance Test-Driven Development 244
Test-Driven Development 246
Automated Acceptance Testing 247
Acceptance != Acceptance Testing 252
Reusing Tests 253
Regression Tests 254
Acceptance Testing Checklist 257
Chapter Retrospective 258

Chapter 8 Effective Collaboration 261

Individuals and Interactions 262
Collocate 263
Set Up a Team Room 266
Meet Effectively 267
Listen Actively 269
Collaborate Productively 270
Be T-shaped 271
Achieve Continuous Feedback 272
Collaborative Development Practices 274
Collective Code Ownership 274
Commenting in Code 277
Associating Commits to Work Items 279

Pairing, Swarming, and Mobbing 280
Branching 286
Chapter Retrospective 290

PART III IMPROVING

Chapter 9 Improving Flow 295
Visualizing Flow 296
The Kanban Board 297
Managing Flow 299
Limiting WIP 301
Managing WIP 302
Inspecting and Adapting Workflow 304
Flow Metrics 306
Calculating Flow Metrics 308
Flow-Based Scrum Events 310
The Sprint 311
Flow-Based Sprint Planning 311
Flow-Based Daily Scrum 314
Flow-Based Sprint Review 317
Flow-Based Sprint Retrospective 318
Chapter Retrospective 319

Chapter 10 Continuous Improvement 321
Common Challenges 322
Impediments 322
Estimation 323
Assessing Progress 326
Renegotiating Scope 331
Undone Work 333
Spikes 338
Fixed-Price Contracts and Scrum 339

Common Dysfunctions 341
Not Getting Done 342
Flaccid Scrum 343
Not Inspecting, Not Adapting 344
Developer Challenges. 346
Working with a Challenging Product Owner 349
Working with Challenging Stakeholders 353
Working with a Challenging Scrum Master 355
Changing Scrum. 358
Becoming a Professional Scrum Team. 360
Get a Coach 361
Build a Cross-Functional Team. 362
Achieve Self-Management 363
Improve Transparency 363
Professional Scrum Developer Training 364
Assess Your Knowledge 365
Become a High-Performance Scrum Team 366
Chapter Retrospective 367

Chapter 11 Scaled Professional Scrum 369

The Nexus Framework 370
Nexus Process Flow 371
Nexus Integration Team. 372
Nexus Events 373
Nexus Artifacts 377
Integrated Increment 379
Nexus Support in Azure DevOps 380
Configuring Additional Teams. 380
Managing the Product Backlog. 385
Chapter Retrospective 386

Index 389

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

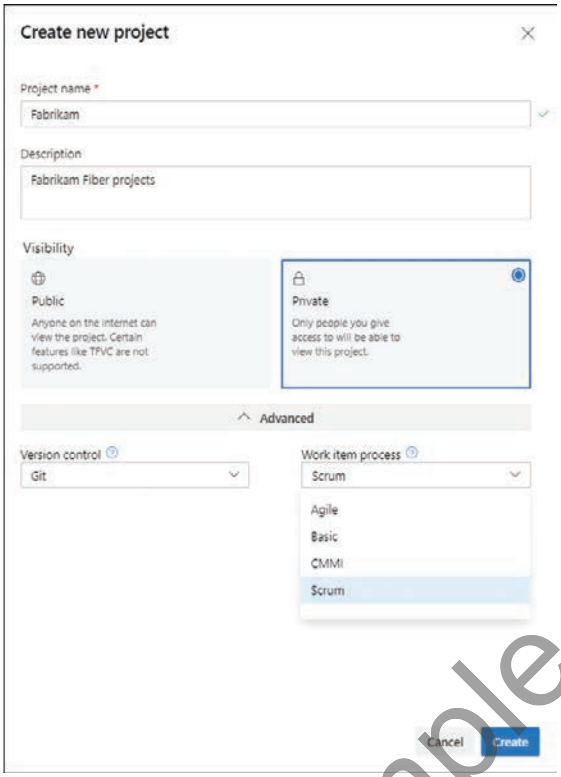
Several processes are available out of the box. These system processes are designed to meet the needs of most teams. Some of them are more formal, like the Capability Maturity Model Integration (CMMI) process. Some of them are lightweight, like the Basic process. Some of them are intended to match the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

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.

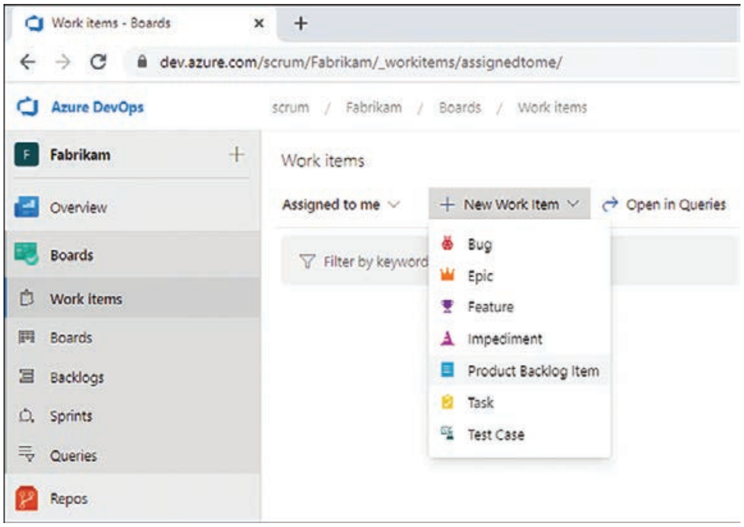


FIGURE 3-2 Work item types available to a Scrum project.

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.

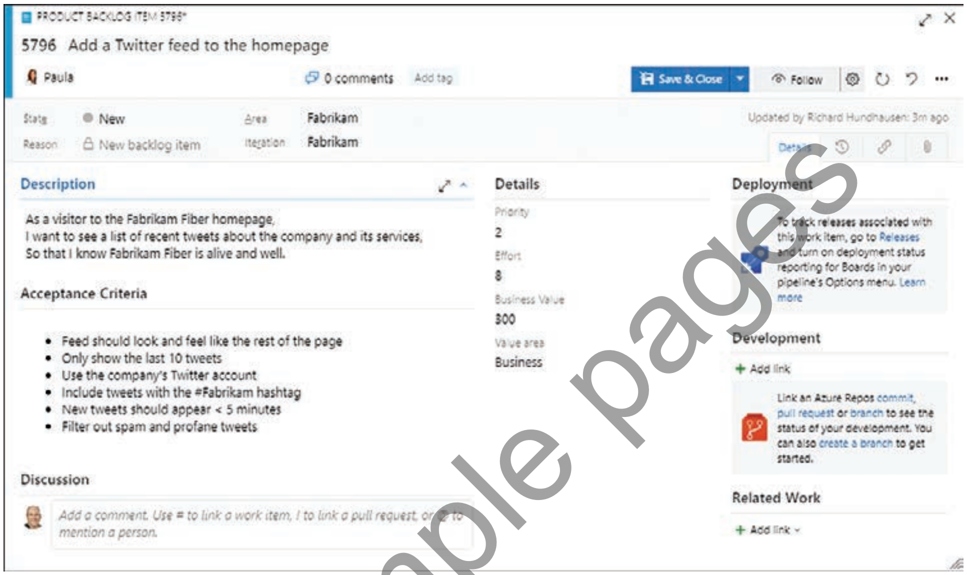
Scrum Work Item Types

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

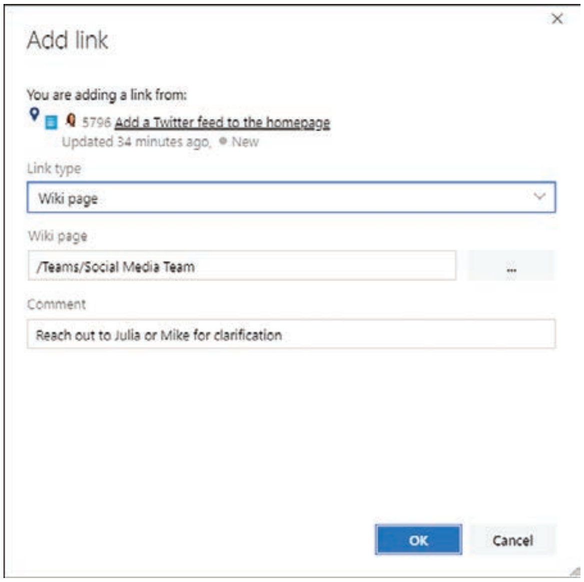
FIGURE 3-3 Adding detail to a PBI work item.

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.

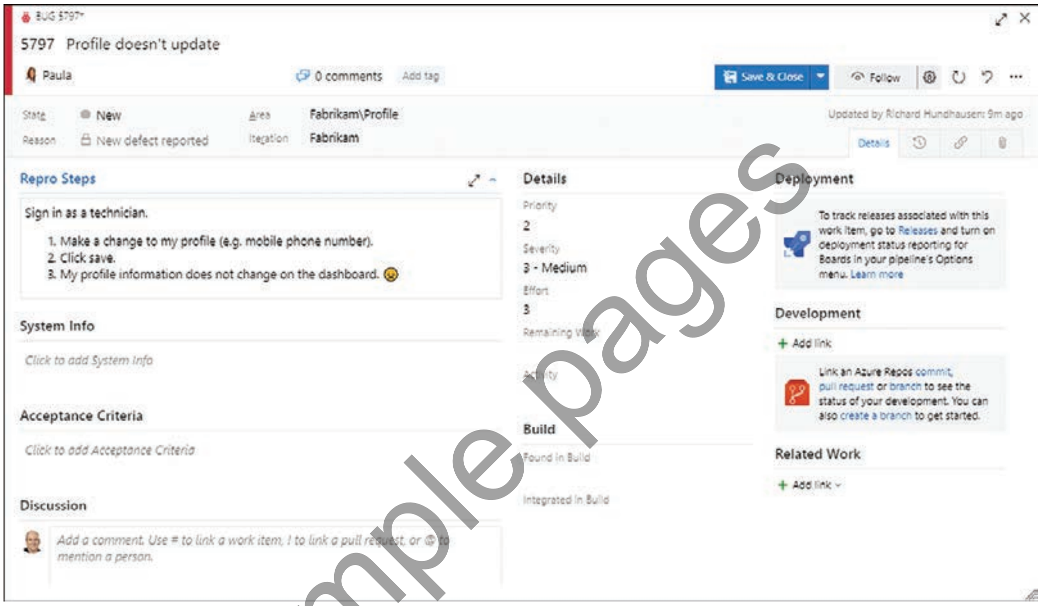
Smell It's a smell when I see tasks created and associated with a PBI prior to Sprint Planning. Perhaps the Scrum Team knows what the plan will be, but what if it changes? The creation and management of those tasks will be wasted time and, what's worse, stubborn Developers may want to stick to their 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 · Area: Fabrikam\Profile

Reason: New defect reported · 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

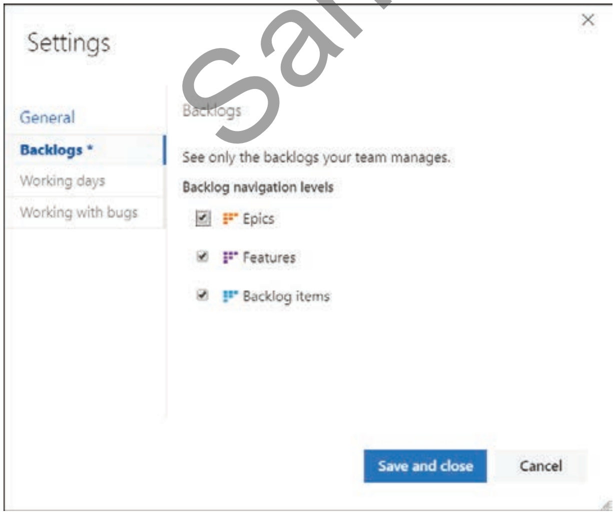
In Scrum, there is only one Product Backlog for a product and it contains only Product Backlog items. Some PBIs are quite small, deliverable in a single Sprint or less. Other PBIs are larger and may take more than one Sprint to complete. Huger PBIs may take many Sprints, even up to a year or more to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


Settings

General
Backlogs *
Working days
Working with bugs

Backlogs

See only the backlogs your team manages.

Backlog navigation levels

☑ Epics
☑ Features
☑ Backlog items

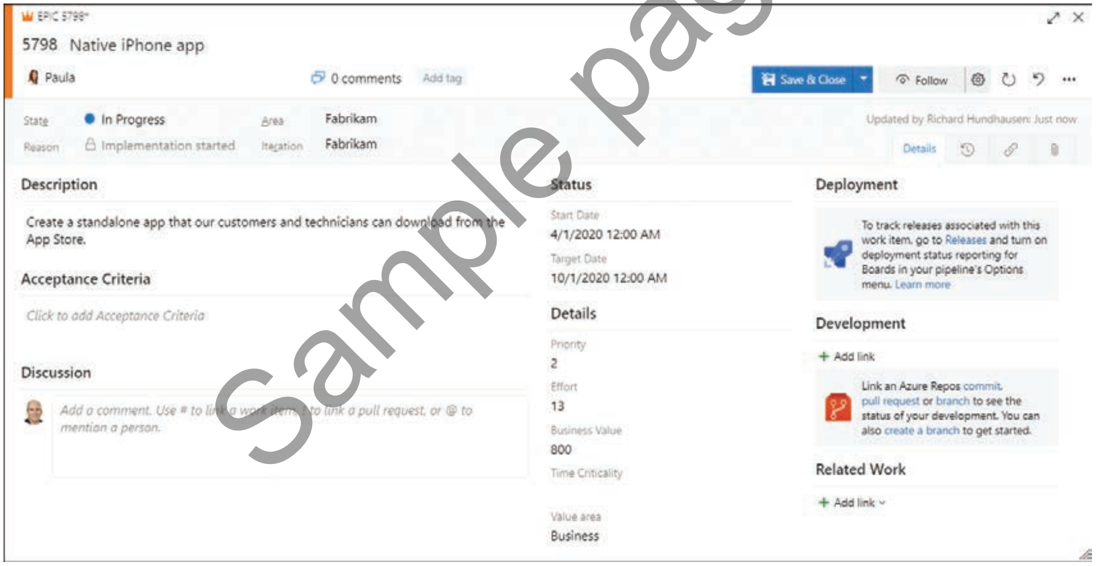
FIGURE 3-6 Enabling the Epics backlog.

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.


EPIC 5798*

5798 Native iPhone app

Paula | 0 comments | Add tag | Save & Close | Follow

State: In Progress | Area: Fabrikam

Reason: Implementation started | Iteration: Fabrikam

Updated by Richard Hundhausen: Just now

Details

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.