AGL Academy

A community effort by Agile government professionals to help educate and empower those who seek to implement Agile processes into their own agencies.

Powered by Agile Government Leadership

By bringing applied Agile practices to government, we want to redefine the culture of local, state and federal public sector service delivery across all aspects of government. We will work with Agile professionals and organizations to support their work in getting Agile infused into government processes. We will foster a spirit of openness and mentor those new to Agile so that they have the necessary practical advice, resources, tools and community support for successful deployment. Through Agile Government Leadership, we will create a responsive, engaged government that more efficiently and effectively serves its citizens.

Connect with AGL

- Website
- Subscribe
- LinkedIn
- Twitter

Agile for the Government Product Owner

Welcome.

This material is designed for the Product Owner in government seeking training and knowledge on how to be effective in an Agile environment. In the Scrum framework for practicing Agile, the Product Owner is the person who represents the business and user community, and is responsible for determining what features will be in the product release. This person is typically the project's key stakeholder and is responsible for maximizing the business value of the product.

Here you will find a step-by-step course to get started. It includes readings, videos, general introductory info, advice specific to government settings, and exercises that will empower you to start providing the benefits of Agile development to your stakeholders.

You have several options for completing this course:

- Use this document (contains links to outside reading material and resources)
- Use our website (http://www.Agilegovleaders.org/academy)

You will work through a series of tasks each week. These start out as simple readings for you to discuss with colleagues, then progress to exercises you must perform that model the Agile process on a small scale to give you hands-on experience. With a little effort and cooperation from Project Managers on your team, in a few weeks you will be ready to begin a serious Agile project.

Before you begin, please note:

- The Product Owner is the person who represents the business or user community and is responsible for working with the user group to determine what features will be in the product release.
- This course is intended for a government Product Owner. It is not aimed at the Project Manager who is managing the development team.

- Although there are many Agile methodologies, this document is about Scrum, a common and standard Agile practice that is widely used.
- As a government professional you may have to deal with procurement and other issues. Although we discuss these challenges, this curriculum teaches you Agile product ownership, not Agile procurement.
- This is an introductory course -- there are many other important topics in Agile development which we will mention at the end when we point you in the direction of the "Next Level".

We at Agile Government Leadership hope this course helps you to wow your citizens with efficient delivery of phenomenal digital services. We welcome and appreciate feedback from you on what could allow us to iteratively improve this curriculum for the next POs who take on this challenge.

Contributors

This course was designed by Agile government professionals with combined experience at the federal, state, and local levels, in addition to the private sector. Contributors to this course are members of the AGL Working Group: Elizabeth Raley, Bill Haight, Tim Nolan, Son Tran, Robert L. Read, Doug Birgfeld, Mark Vogelgesang, and Joshua Smith.

Lesson 1: Introduction To Agile

GOAL: By the end of this lesson and after completing all the items below, you should have a basic understanding of Agile and Scrum, along with the motivations for using them in government projects.

What Is Agile?

Agile is an iterative and incremental method of managing and developing projects with a team. It focuses on customer collaboration, responding to change, and frequently releasing working software for user feedback.

The Agile Manifesto is a single, simple, brilliant sentence that you should refer to often:

> "Through this work we have come to value:
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation
> - Customer collaboration over contract negotiation
> - Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more."

Let's take the manifesto and put it into the context of real work, real customers, and the real world. Agile takes the best of the world's history of getting things done and rolls it into a single framework. For instance, we know that people work better in teams; we know that trusting relationships has better results than relying on legal documents; we know that predicting the future is super hard; we know that the proof is in the pudding. We also know that 3 things 100% done are more useful than 10 things 75% done. And finally, we all know the phrases "one day at a time", "one game at a time", and "keep your eye on the ball".

Agile takes these intuitive truths and brings them into the world of development, software, and beyond. Agile is not a prescriptive dogma. It is a set of activities and methods that can be used individually to make things better, and collectively to make things great.

Agile takes practice and disciple. But just like that play you were in, or the concert you practiced for, you will be happy with the ovation at the end. It's worth it.

Why Agile in Government?

The Agile Manifesto follows 12 Principles that focus on value, collaboration, motivated people, change, rapid delivery, and simplicity. These same principles apply to government regardless of whether your agency is Federal, State or Local. The principles offer a pragmatic approach for working with customers, citizens, employees and each other.

An Agile government provides excellent services and value because managers trust their teams to do the work and to collaborate directly with customers. Instead of micromanaging and frustration, there is daily communication and collaboration. This culture of transparency rubs off on the customers within an Agile government -- and soon, the culture of the agency begins to shift.

With Agile processes paving the way for delighted customers, a government's focus can be on providing genuine happiness instead of settling for "good enough" or shrugging off another failed IT project. Eventually, a government practicing Agile will adopt a permanent Agile mindset. The 12 Principles behind the Agile Manifesto will become self-evident instead of being something to strive for.

The easiest way to identify and explain what Agile offers to your agency is to replace it with "flexible". We want our government to be flexible to meet the changing needs of our customers and citizens. We want our government to be flexible to produce the greatest value within the allotted time frame and budget.

Being Agile (flexible, transparent, efficient) should be the goal of every government agency.

Reading List

Study each of these resources and discuss what you're learning with a colleague.

Agile Government Handbook

AGL has compiled a handbook that outlines Agile in the government space.
Study Time: 90 minutes.

Government Challenges to Using Agile

This article discusses government-specific challenges that you may face in moving your team to Agile methods.
Study Time: 10 minutes.

The Scrum Guide

Scrum is a framework for practicing Agile. This guide lays out the Scrum roadmap and will serve in a full understanding of how to practice Scrum.
Study Time: 30 minutes.

Agile Terms

New vocabulary will come up while talking about Agile and Scrum. This is a cheat sheet to ensure the team has a shared understanding of what is being discussed.
Study Time: 20 minutes.

Video List

Watch these videos to gain a deeper understanding of the how-to behind Agile and Scrum. Continue to discuss your findings with team members to ensure that you understand the concepts.

Agile 101 Case Study Discussion

Study Time: 1 hour

Scrum Training Series

(Individual videos below) Study Time: 3 hours

Scrum is a framework for Agile that addresses many of the issues that we face daily in website development: changing business needs, changing technology, high value needs, and more. It provides us with a roadmap to follow in order to run an Agile project.

- Intro to Scrum
- Backlog Refinement
- Sprint Planning Meeting
- Daily Scrum Meeting
- Sprint Review Meeting
- Sprint Retrospective

Agile Assessment

Determine your team's current Agile capabilities using this Agile Assessment. This will help you see how Agile will be supported or challenged by your agency's leadership, culture, policies, etc.

CONGRATULATIONS, YOU'VE COMPLETED LESSON 1!

Lesson 2: How to be a Product Owner

What is a Product Owner?

In the Scrum framework for practicing Agile, the Product Owner is the person who represents the business and user community, and is responsible for determining what features will be in the product release. This person is typically a project's key stakeholder and is responsible for maximizing the business value of the product.

Serving as a proxy for the end-user or customer, the Product Owner is responsible for maintaining a vision of what he or she wishes to build, and conveying that vision to the Scrum team. This is essential to successfully starting any Agile software development project.

What Makes a Good PO

Product Owners can come from a variety of backgrounds, ideally bringing experiences and relationships that have prepared them to think like their customers (users). Experience in customer service, marketing, product development, and business process management will give Product Owners a head start in performing the role successfully.

People who have a talent for leading groups, strong interpersonal and listening skills, and above average emotional intelligence will also do well in the role. Since the PO will manage communications among organizational stakeholders, pre-existing relationships up and down the organization are a huge benefit to new POs.

While specific subject matter expertise can be helpful, the most essential trait of a good PO is a willingness to be curious and engaged in learning about the users that the product is meant to help or serve. POs do not need technological experience, but it is helpful if they have an interest in technology as a tool and recognize its importance to the project.

What Does a Product Owner Do?

The Product Owner maximizes the value of the product and the work of the Development Team by owning specific operational responsibilities within the Agile Scrum team. It is important that the PO fulfill the responsibilities for their own role without overstepping into other roles.

Responsibilities Facing Inward to the Agile Team

- The PO must articulate the overall vision and priorities for the project. They do not necessarily have to write user stories, but they must make sure that the user stories represent what the user really wants.
- The PO must be able to prioritize work. Presented with a variety of conflicting goals and opinions, they must make the final call for hard product decisions.
- The PO is responsible for making trade-offs between features and effort in order to maximize the value of the product and the work of the Development Team.
- The PO must require a burndown chart and use it to predict the progress of the team in order to make good priority decisions and to communicate progress to stakeholders.
- The PO approves or accepts completed work (or gains sign-off by others) upon completion of acceptance criteria that has been previously agreed upon.
- The PO is not expected to have technical expertise. However, they do have to understand the Agile process enough to get straight answers from the Development Team and to treat the developers as equal peers.

Responsibilities Facing Outward to the Rest of the World

- The PO serves as a conduit carrying communications to and from the Scrum team and acting as liaison with upper management and other departments that have a bearing on the project's success. Part of this responsibility is managing stakeholder expectations.
- The PO must manage relationships with stakeholders to make it possible to discover exactly what the user most needs. The stakeholders include executives, but the end-user is the most important.
- The PO must understand the user, and this means understanding approaches and technologies for learning about the user. These include user interviews, surveys, design studios, and other user experience design approaches.

- The PO ensures that features developed for the website (or other product) meet Key Performance Indicators (KPI) or forward key business initiatives. They keep their eye on the ball, and keep the organization's resources focused on the most important things.

Product Ownership in the Government Context

Being a PO in the government context presents some unique challenges.

Day Job

The largest challenge is often that as a Product Owner in the government sphere, you also have a "day job", meaning that you also have a full slate of normal duties and responsibilities in addition to serving as a PO for your department's project. It is very important that your time as an acting PO is supported by leadership and that you are allowed the time necessary to fully engage in the project you are working on.

Too Many Stakeholders

In theory, the PO is the liaison between the Agile team and the rest of the organization. Nobody but the PO should set the direction for the Agile team. When that organization is an entire state or national government, it is unrealistic for the PO to represent the millions of organizational stakeholders with 100% accuracy. This is a problem in all organizations, but is exacerbated because of the size and federated nature of government organizations. Untangling and de-conflicting all stakeholder priorities is not possible. The only answer for the government PO is to use their best judgement, demonstrating leadership amid ambiguity.

Legacy Process Drag

This means that you may be faced with stakeholders who are very attached to the way things are now, and to the historical or mythological reasons as to why things can't and shouldn't change. Legacy aligned stakeholders often have seniority or other levers of influence that can slow or disrupt the clear articulation of a single focused set of priorities. It will be the PO's job to find a way to address this barrier.

Political Priority Soup

This is the circumstance in which a project may serve multiple departments or sub-bureaus, each of which will have a leader who has an agenda. It is sometimes tempting to allow each sub-group to set their own priorities. This can lead to having more than one top priority, since each sub-group will give the impression that their priority is the most important. If there exists - in fact or assumption - more than one top priority, there will be tears. Consensus **must** be made about a single set of priorities with a single #1 at the top. It is best to set those with the collective from the get go.

Name Dropping
This can happen anywhere, but has a rich tradition in the government context. This is the circumstance in which the team is setting priorities or finalizing a workflow, and a stakeholder utters, "I am glad we came to consensus on this but it's *really important to the **Commissioner** that we do what I want."* This is an example of an artifice that can create confusion and consternation instead of a satisfactory conclusion.

This practice can be averted by creating ground rules that discourage informal decision making. While informal discussions amongst the team and leadership is mostly a good and natural strategy for solving problems, it can become troublesome when big decisions that impact the team are made this way. As a PO you will work closely with the Scrum Master or Project Manager to create a communication and decision making scheme that execs and team members can follow without the need to invoke the names of higher powers

How the Product Owner Relates to Other Roles

How the PO Relates to the Customer

The Product Owner's chief responsibility is to represent the needs and desires of the customer or end-user to the rest of the Agile team. The PO gives the team a single point of contact, providing regular updates and clarifications to the central question "What are we trying to build?"
The PO must first discover what the population of customers want. In practice, one customer may want the Agile team to build something different than what another wants. Or a dozen

customers may want a dozen different things. The PO must use experience and judgement to pull together all the customer voices into a single, coherent message to the Agile team.

How the PO Relates to Government Executives

Just as they must represent the needs and desires of customers, the Product Owner must also represent the needs and desires of executives and other internal organizational stakeholders to the Agile team. By communicating through the PO, executives provide inputs to the team and receive outputs from them, such as demos, briefings and reports.

In addition to direct inputs, executives shape the organizational environment within which the Agile team works. The organizational capability granted by executives to the team will have an impact on the PO's answer to that central question "What are we trying to build?" Just because a customer wants the team to build a certain something, doesn't mean the team has the organizational capability (i.e., budget, staffing, strategic alignment, etc.) necessary to build it. The PO is responsible for figuring out what the customers want, balancing that with the organizational capability granted by executives to the Agile team, and finally communicating the resulting product direction to the rest of the team.

How the PO Relates to the Scrum Master

The Product Owner and the Scrum Master overlap in some of their skill sets, which means the individuals filling them should be able to collaborate well as peers. They both play leadership roles within the Agile team, but the PO typically has greater authority in the organization.

Several specific areas are well suited for collaboration between the two:

Team Communication and Morale
Working closely with the Scrum Master, who works directly with the team every day, the PO can make sure all necessary communications regarding the project are clear to the team. The PO and Scrum Master can keep a team's morale and productivity at the highest levels simply by keeping lines of communication open and setting clear priorities.

Managing Dependencies With Other Teams

Both the Product Owner and Scrum Master likely have close relationships with other teams, and may be in contact with past team members that work elsewhere in the organization. Leveraging those relationships can help to improve the current project.

Clarifying the Product Vision

While the PO is responsible for establishing and communicating the strategic vision behind the product, the Scrum Master plays a role in bringing the team on board and making that vision a reality. It makes sense for both to be fully engaged in the vision and be able to communicate it appropriately and clearly.

The Product Owner and Scrum Master roles also serve as a check and balance to each other. For example: the Product Owner may prioritize work, but the sprint work but must fit within the capacity allowed by the Scrum Master. Another example: the Scrum Master encourages the team to rapidly accomplish work, but sprint points are not earned until the Product Owner approves the quality of the work. An imperfect - but useful - analogy from traditional project management methods is the role relationship between a project manager focused on keeping to the schedule and a business analyst focused on preserving quality and value.

How the PO relates to the Development Team

The Development Team plays a critical role in bringing the Product Owners' vision into reality. Beyond setting the vision, the PO must also collaborate with the Development Team to prioritize work, clarify requirements, and review quality of outputs.

Unlike some other project management methods, Agile management gives the Development Team exclusive responsibility to estimate the complexity and effort a product feature will require. The PO has no right to change or apply pressure to change the Development Team's estimate. If initial estimates are higher (or lower) than expected, the PO may work with the Development Team to identify trade-offs in feature requirements that may cause an update to the estimate.

How Product Ownership Differs from Project Management

Product Ownership is about representing the end-user, customer, and other stakeholders. Project Management is about managing the Development Team to realize the vision of the Product Owner.

In Scrum, the role of the Product Owner and of the Project Manager (or Scrum Master) are completely separate roles filled by two different people. Each role requires the complete focus of a single individual, and more importantly, the roles are in healthy conflict with each other. A Scrum Master should be agnostically removing blockers for the team with no opinion about customer-set priorities; the Product Owner specifically represents the priorities of the customer with no opinion on how the team meets those priorities. It is literally and truly impossible for the Scrum Master and Product Owner positions to be filled successfully by one person.

Common PO Challenges

How do I prioritize features when it's ALL really important?
Remember the 80/20 rule. Studies confirm that 80% of customers use 20% of the features of the product. Pick user stories that affect the most people or have the most value to the customer.

There is often the temptation during a development project to try to identify every possible thing that could ever be needed for the conceivable future. It is natural to want to take the biggest bite of the apple when it is offered. However, trying to predict everything that could ever be needed is nearly impossible. Things change too fast.

The solution is to focus on the parts of the system that are super critical to the organization or the customer. It may be that you can identify one feature - such as an intake workflow - that is used by every customer and is critical to the business, while another feature - such as a change of address workflow - presents a cumbersome manual process to customers. It is tempting to focus on fixing that manual process to make life easier for customers, but ultimately the intake workflow should be given the higher priority because it is absolutely essential.

I report to executives who think we've bought "x" . . . so when priorities and timelines inevitably change, how do I explain that things have come up, taken longer, etc?
Develop a habit of communicating simply, clearly, and often with your executives so everyone feels like they know what's going on. Write into the project plan that changes will occur and will be communicated.

Nobody likes surprises when it comes to project status. From the beginning, set an expectation that there will be changes throughout the project lifecycle. When changes occur, talk openly about what you've learned that prompted the changes.

One way to do this might be to have a standing, short, biweekly meeting with execs - similar to a stand-up meeting - where the PO communicates progress, challenges, and changes. This way the execs have some visibility into the process and can lend assistance in addressing challenges early and often. This will also increase the executives' trust in the PO and the Agile process.

If we have a fixed budget and a deadline. How do I effectively manage scope so that we have something launchable on time?
Select user stories that all support a single product line or workflow. It's better to have the release be a fully functional improvement than a bunch of unrelated stories from multiple product lines or workflows.

Part of this is to be honest about how many features you really need in order to achieve goals. Focus on the minimal set of features that make the product viable and usable (referred to as the "Minimum Viable Product" or MVP). You can go back later to buff them up.

The other part is to be disciplined and forceful with stakeholders, resisting the temptation to try to please everyone by prioritizing little pieces of everybody's individual favorite feature set. A good way to get on the right foot is to prioritize the Epics into an absolute order.

Imagine that there is a project to improve a case management system. For this project the customer, working with the PO and and other stakeholders, has identified a number of Epics that need to be improved or created in order to make the case management system function optimally. All these things must be built for the system to work well. The Epics are listed as:

1. Intake
2. Vetting

3. Approvals
4. Payments
5. Enforcement

By prioritizing these epics in ranked order, the developers and rest of the team can focus on one Epic at time. Instead of doing a few stories from Epic 1 and a few from Epic 3 and some from Epic 5, the team should focus on one at a time so that a completed area of business function can be released as soon as possible.

This prioritization exercise should include all the stakeholders if possible, being revisited periodically to allow for changes if needed. But during a sprint the teams should focus on stories from the highest priority Epic.

How do I instruct the developers in the creation of features if I don't know anything about software design?
The great thing is that you don't have to, and the Devs really are happy to do this part. Just be clear about your goals and what you need - they will take care of the design. Tell the team what outcomes you want the product to achieve, rather than trying to articulate functional requirements.

For instance, you may have a legacy system that you have been using for intake. As you redevelop that workflow, it may be tempting to tell the developers to design the new system to look and function like the old system, because you are comfortable and familiar with it. But the old system might have collected all the info on one long screen and used a lot of "jargon".

The user story says that you need to collect intake information in a way that reduces data errors and encourages people to complete all the optional fields. It is not appropriate to insist that the developers make it like the old one, because the UX expert on the team knows that a Wizard tool for intake that uses plain language and steps the user through 4 short pages will satisfy the story more effectively. There is no need for you to be the expert, but you need to trust and feel comfortable with the experts you have.

How can I tell if the developers are being unreasonable?
You may experience push-back from the Development Team if they are interpreting a given requirement differently than what you had envisioned. If they respond that a requirement is too large or unachievable, it's time to have a clarifying conversation.

Ask the development team to explain the requirement to you in their own words, using business language rather than technical jargon. This will often identify where their interpretation of the requirement has diverged from your own. If developers seem unreasonable, look for these three common causes:

1. They think they are not being understood
2. They were surprised by something they thought was all set
3. They think you are telling them how to do their job (see previous section)

These issues are avoided by having frequent contact with the Development Team and making sure everyone is in communication from within their role. For instance, it may seem like developers are being unreasonable if they say something like, “You don’t need that feature.” But it could be that what they are really saying is, “In order to get the business outcome you want, you don’t need to build the thing you are asking to build.”

Typically, these conflicts come down to a misunderstanding and can be worked out by having the Devs explain to you how your outcome will be achieved. It may be that they misunderstood the outcome you are seeking. This interaction gives you a chance to listen to their ideas and correct any misunderstandings that could be present. Developers are there to come up with the “how”; you are there to be clear about the “what”. If their “how” gets your “what” then the outcome will be success.

How can I demand a clearer explanation from engineers?
You can remove the need for having to “demand” anything during the project by working closely with the whole team and utilizing the Agile process to keep lines of communication open and smooth.

This is where the concept of Habituation shows its value. Habituation involves using the procedural scaffolding of the Agile process to allow the whole team to focus on the context and content of the work. By consistently using activities like daily standups, sprint planning, sprint review, and retrospective the circumstance of needing to demand anything is removed - you will get the information you need through the intentional use of these activities.

In the example above, where a PO feels they are not getting a straight answer from the Devs, it helps to start with the end result. Did this lack of a straight answer result in a failed story or other barrier to the team? Even if it didn’t, the feelings of miscommunication is causing bad feelings and distrust to grow within the PO. The best way to handle it is to speak to the Scrum

Master and work together to find a solution. The Scrum Master may bring the issue up during a retrospective and will likely work with you and the Dev Team to remove the barrier. It's best that the whole team be part of the solution.

How should I be transparent about potential changes?
As you receive functional components of the product, new ideas and stories will inevitably develop, which should be captured and put on a backlog. The Agile framework naturally allows for these "changes" - which are really refinements, discoveries, and improvements - to be addressed in the normal cycle.

The best way to communicate changes is in the context of the business outcomes that stakeholders are seeking. For example, one of the goals for a project may be to give customers better access to information about their case. The team decides that they need a phone line, an IVR, with a call menu tree. At first it's decided that there should be two choices for callers:

1. Press one for status of case
2. Press two to open a new case

After the team develops and releases the call tree, it works so well that someone suggests adding two sub-choices under option 1:

1. Press one for status of case
    a. Press one for cases in the south
    b. Press two for cases in the north

The new stories are added to the backlog. During sprint planning, it is discovered that 80% of cases are in the south, and the south is made up of 4 regions. Since the north has just one region, the options are restructured by region:

1. Press one for region 1
2. Press two for region 2
3. Etc.

The Agile process demands that the PO return to stakeholders and make sure this new feature fits the bill. If it does, the PO changes the story in the backlog and it is added to the next sprint. A change of this size can be left to the PO. If the change results in a conflict that can't be resolved

without executive involvement or it risks adding significant cost, the change is escalated to be addressed in a regular meeting with execs about status, challenges and changes.

Why should I want the team to build a Minimal Viable Product (MVP)? How do I convince my stakeholders to accept this approach?
Building to the Minimum Viable Product provides early and long-term benefits to both the developers and the customer. Help your stakeholders and users focus on what they really need for the best business process, rather than allowing them to focus on lists of features that are difficult to achieve with MVP.

Here are the key benefits of an MVP:

- Quickly provides a working product to end users that tangibly improves their life and work
- Customers can withdraw the invested value of the effort of development right away
- Decisions about improvements can be made in a real world context
- Developers can focus on aspects of the system that bring the highest value to customers
- The organizational change of implementing a new system is less disruptive, more inclusive
- Everyone gets to practice and refine the dev-build-release process in low risk scenarios

Here is a sample storyline of how a new development project (and the concept of incremental development) is experienced in the government context.

Project Team: (at a kick off meeting) "We are going to redevelop or make a new case management system because, (reasons)."
Stakeholders: (to themselves) "Oh no." (to the project team) "The last time we did this, woe and despair fell upon us, we never really finished it, and there are lots of things we still need."
Project Team: "It will be different this time - we have this Agile thing we are doing."
Stakeholders: "Okay… Here is a list of everything we ever wanted, prioritized by the stuff you guys missed last time."
Project Team: "Actually, we want to focus on just the basic bits you really need, then after we release that, we'll keep improving it."
Stakeholders: "Look, if we don't ask for everything, we will get nothing - plus, we need a full detailed plan for the grant, so this incremental approach just won't work here."

It is important, when talking about MVP, not to frame things in terms of what actual features will be developed. As evidenced by cable company packages and smartphones, people will spend a lot of money and time on a list of features they don't really need, simply because a longer list feels more impressive and useful. Instead, focus the discussion on the real world workflows and scenarios users will encounter - the actions they will take and how they will use the product.

The best way to direct stakeholders' energy into a positive circuit is to collaborate with them on how to make the business process better, and then ask the questions:

"What needs to change about the system to make the new business process usable?"
"What changes do we need right now, and what changes would be helpful later?"

When users have new ideas, they want to see those ideas come to life. MVP allows your team to quickly show the users how their ideas work in actuality, test and refine them, and make everyone's life a little better in the process.

When introducing the concept of MVP, make your customers comfortable by explaining the process clearly, and give the group a chance to practice on a low-risk, high-reward deliverable, such as the IVR system example above. The team was able to convince the stakeholder to use the MVP because they targeted a real pain point, and offered a way to relieve that pain right away. They also promised to keep refining it if needed, and they kept the promise.

It is super important that everyone keep their promises. The PO, representing the customer/user, promises to keep the product as simple as possible, while the developers promise to make it only as complex as it needs to be to solve the problem.

How do I effectively manage the expectations of my various stakeholders?
Effectively managing stakeholder expectations takes strategic thinking, great communication, and lots of practice. One of the best ways to correctly set expectations from the outset is by educating your stakeholders on the product vision and the Agile process.

The Product Owner must constantly articulate the product vision -- the unifying theme that all stakeholders can relate to, regardless of their specific role inside or outside the organization. In

addition to the vision communicating an outline of what the product WILL be, the vision should communicate what the product WILL NOT be. This helps to keep expectations streamlined.

The Product Owner can also set correct expectations by educating stakeholders on the Agile process. Throughout the course of developing a product, it is nearly certain that plans and priorities will change. The core Agile process does not need to change. Managing expectations becomes much easier if your stakeholders can focus on and understand the process rather than focusing on specific outputs.

What should I do when things don't go as expected?
It is expected that things won't go as expected. The Agile methodology assumes there will be changes to the initial plan, and includes a process for dealing with these changes productively.

If the Product Owner has educated stakeholders about the Agile process and its ability to deal with change, there should not be major upheaval when things don't go as initially expected. The process itself does not change - rather, the discussions and decisions that happen in meetings are free to reflect the current reality (rather than pretending the initial expectations have held true). This honesty allows the team to make the best decisions for improvement going forward.

If your stakeholders, boss, or team are having trouble riding the waves of change, point them back to the value of using change to create a better end product. Remind them that being Agile means being flexible.

How do I push back on high estimates from the Development Team?
The Developers are the experts on estimating the complexity and effort required to build the features described by the user story acceptance criteria. Rather than pushing back on high estimates, probe the team to better understand potential changes in the acceptance criteria that would result in lower effort.

Often the Product Owner is surprised by how high the development team's estimates are for effort and complexity of making the product vision into reality. In that case, it is not productive for the PO to question their expertise. Instead, make sure you understand which aspects of the requirements or user story acceptance criteria are driving the effort and complexity. In many cases, the estimate can be reduced by removing high-effort, low-value acceptance criteria from the user story for that feature.

Reading List

Study each of these resources and discuss what you're learning with a colleague.

Scrum Alliance CSPO Learning Objectives

The Scrum Alliance offers a Certified Scrum Product Owner credential to individuals who, among other qualifications, have been trained to meet these learning objectives.
Study Time: 30 minutes.

U.S. Digital Services Playbook

The U.S. Federal government created a playbook of 13 key "plays" drawn from successful practices from the private sector and government that, if followed together, will help government build effective digital services. While not exclusively relevant to the Product Owner, this resource provides a broader context for what should be happening in the project.
Study Time: 30 minutes.

The TechFAR Handbook

The TechFAR Handbook highlights the flexibilities in the U.S. Government Federal Acquisition Regulation (FAR) that can help agencies implement "plays" from the U.S. Digital Services Playbook that would be accomplished with acquisition support. While not exclusively relevant to the Product Owner, this resource provides a broader context for procurement activities leading up to the project.
Study Time: 60 minutes.

Video List

Watch these videos to gain a deeper understanding of the how-to behind the Product Owner role. Continue to discuss your findings with team members to ensure that you understand the concepts.

Agile Project Ownership in a Nutshell

This is basically a 1 day PO course compressed into a 15 minute animated presentation. There's obviously more to product ownership than this, so remember this is a high level summary.
Study Time: 16 minutes.

The Essential Product Owner - Partnering with the Team

Even the best POs struggle to meet the demands of their "regular business-focused job" while providing sufficient team guidance. Agile expert Bob Galen shares real-world situations where he's observed product owners who deliver truly balanced value for their business stakeholders.
Study Time: 75 minutes.

Review

(coming soon)

Now take the PO Self Assessment to evaluate yourself on your ability to be a Product Owner. Be honest with your capacity; you can take this assessment again later to see how it's changed. Certainly after your first project as a Product Owner you will see a change in some of your responses, as practice makes a big difference in your capabilities.

CONGRATULATIONS, YOU'VE COMPLETED LESSON 2!

Lesson 3: Creating and Managing a Backlog of User Stories

Exercise 1: Writing User Stories

Stories are the building blocks of Agile projects and represent the fundamental unit of communication and tracking progress.

- **Estimated Time:** 60 minutes
- **Materials Needed:** Index cards or Post-It notes, pens
- **Outcome:** You know the standard story format and can a write a good user story.

Preparation

One of the most important aspects of moving to Agile is understanding "stories". It takes practice to write good stories, and this exercise allows you this practice. As the Product Owner, you must deliver your customer's or stakeholder's perspective and share with the project team what is needed and why.

A user story must provide value to some user. An Agile process is driven by the completion of stories, each of which provides tangible, demonstrable value to the user/customer/stakeholder. A sprint consists of a set of conscientiously prioritized stories. Experience will show that it's best to use a format for each story that identifies **who** the user is, **what** they need, and for what purpose (the **why**). Such stories are written in this format:

"As a ___, I need a ___ in order to ___".

The **who** in a user story could be someone with a particular functional role, who holds a certain title, comes from the perspective of a persona, or embodies the needs and behaviors of a hypothetical user.

The **what** in a user story details in specific terms the need, feature, or functionality desired by the **who**. This is what your project team will build into the product or service.

The **why** in a user story states the value. It presents the needs of your users and customers up front and center.

Here's an example of a user story that clearly defines the **who**, **what**, and **why**: "As a jazz fan, I need a tuning knob in order to find a jazz station on the radio that I will enjoy listening to."

Keys to a Valuable User Story

- Product Owners must have courage to ask for what they believe their users/customers/stakeholders really want.
- A story must have value to someone. It must make the product better in some way.
    - The story when complete will make a real-world task faster, better, easier to understand, have fewer steps, or collect better info.
    - The high priority stories affect the most users or procure the highest value data.
    - Avoid exotic/one-off stories (i.e. edge cases).
- "Clean up the bugs we introduced in the last sprint" is NOT a user story because it does not add anything to the product.
- Remember the INVEST model! Good user stories are:
    - **I**ndependent
    - **N**egotiable
    - **V**aluable
    - **E**stimable
    - **S**mall and
    - **T**estable

How to make your User Stories INVEST-able

You may hear your Project Manager or Development Team assigning "relative points" to user stories. They are describing (or "estimating") the level of effort that a story will require to be completed. Stories with low numbers or small levels of effort will be completed more quickly

than stories with high numbers or large levels of effort. Stay consistent with whatever system your team is using to estimate the stories, as this will help you manage the product backlog better.

If you end up with more than a few stories that will take large levels of effort, you should split those stories into smaller ones. This will be helpful in two ways:

1. Splitting stories into smaller parts can help you identify and abandon any stories with "low-value functionality" -- they require too much effort for their low yield benefit.
2. When you have stories that are equally sized with smaller point values, it will be easier for you as the Product Owner to prioritize the parts of a functionality separately. This comes in handy later when you start to manage the product backlog.

Here are some tips from Agile coach and guru Richard Lawrence on splitting large stories into smaller ones:

- If there are workflow steps to your story, break down the middle steps into their own stories. Example: "I want a CMS that will allow me to post a new article on the agency's main website." There are several middle steps required to achieve this goal, such as "Post article to a staging site," "Get approval from editorial," "Migrate article from staging to production." Make each of these into their own story.
- Different **business rules** should be different user stories. Since business rules often end up defining the functional specs for a feature, they should be parsed out as their own story.
- If you have a story that states that it wants to achieve A with methods "X, Y, and Z" you have a story with an inherent dependency. Oftentimes the story assumes that the first method in the sequence (X) will involve the most effort so Y and Z can be tacked on with minimal effort. But what happens if your boss tells you to change it up and make sure Z works before X does? This will change your estimation on the levels of **major effort**! In such a scenario, it's better to write the story in one of two ways:
    a) It can achieve A with only one method.
    b) After confirming that it can do A with one method, it can do A with all of them.

- Break down a **complex story into simpler ones**. Example: “I want to search for meetings between office A and office B”. The what-ifs of this story will add up quickly for different use cases. Turn those use cases into their own user stories with their own distinct acceptance criteria (e.g. search for meetings with a certain number of attendees, by location, dates, etc.)
- Be willing to change as **new data** and information are obtained. First come to an agreement as to what exactly “good enough” is for the simplest thing you can build right now -- you can add high-value features later on. As you become more informed, split new stories as the new data rolls in.
- A complex **user interface** can really complicate a user story. Start off with a story for the simplest UI and split new stories for fancier, more useful UI.
- **Time matters**. Split a story between a “make it work” use case and “make it fast” use case.
- Sometimes the implementation of a story is not well understood. Break the story into two phases: an **investigation** and the **implementation**. The acceptance criteria for the investigation phase should be questions you need answered. Only do enough investigation to answer the questions -- then learn from it, build something, and proceed from there.

Workflow

Grab a colleague (or several) and use a current project or directive (or choose one from the list below). As PO, you want to be able to communicate to the development team what users need. You don't need to be a technical person to do this, you just need to know what a user wants and why.

These user stories will provide the team with starting points to discuss how they might accomplish something. Instead of saying "I need an event calendar" you might say "A user needs to be notified of upcoming events that are related to topics of interest in her user profile so that she engages with the community about things that matter to her." This story is more descriptive and gives the team a better understanding of the goal so they can base their solution on intended outcomes.

Believe it or not, one of the critical technologies for this exercise is either index cards (3x5 or 4x6) or Post-It notes. Part of the reason you use paper technology is so you can easily move stories around, reorganize and reprioritize them, and throw them away when done. The small size of the cards and notes ensures that you will not write too much into each story.

A story is a promise to have a conversation later between the end-user and developers. Your goal in writing stories is not to work out details, but to discover the most important goals for your project and to organize a project into discrete, testable chunks.

Potential project goals that will help you practice story-writing

- Clean out the garage
- Develop a static website that informs people about dietary impact on breast cancer
- Develop an app for playing "Conway's Game of Life"
- Develop an app for playing tic-tac-toe
- Write a how-to guide for planting a garden specific to your locality

Choose a project goal for the workshop from the list above. If you have a project or directive in mind, feel free to use that.

Activities

Everyone writes stories for 15 minutes that will advance the project goal.
As new stories are written, you may discover ways to improve previously written stories. You may realize that many small, specific stories can be rolled up into a bigger story; or a big story might be split into two or three pieces.

During this time, stories can be rewritten or reworded to make each story as self-contained as possible. This means that a developer should be able to read the story, understand what a user is hoping to do, and create a feature that enables the user to do it. Additionally, a self-contained

story should be understandable when read back to the user, without a lot of explanation needed.

The stories are prioritized into an absolute order.
There can only be one #1, with the highest priority at the top. There are no ties; a specific order must be chosen. Open discussion is allowed, but in the end, you as the Product Owner have authority to set the actual priorities. Things might change as you discover new information, and the backlog can be re-ordered -- but at any one time it is an absolute ranking.

Check the format of the user stories.
Remember to keep them in the correct format: "As a ___, I need a ___ in order to ___".

Think about grouping the stories into sprints.
Now that your stories are prioritized, do you see how you might group them into sprints? The grouping is usually done by the project team, but it's a good exercise for you to review now to get an idea of how complete your stories are and to recognize if there are gaps.

Outcomes

- It may feel overwhelming to create user stories to describe everything you are imagining for this project. Start from where you are and remember your product backlog will continue to grow throughout the life of the project.
- You will get better at creating user stories over time, but this exercise should produce enough user stories that a sprint could be planned. If not, keep trying.

Exercise 2: Backlog Refinement

This exercise will help you gain an understanding of the Product Backlog along with your role in managing it.

- **Estimated Time:** 90 minutes
- **Materials Needed:** Project-tracking technology of your choice (sticky notes will do, but we recommend you try Trello.com, explained in this lesson)
- **Outcome:** You can prioritize a list of user stories into a backlog of work to be done.

Preparation

What is a Product Backlog?

The Product Backlog is simply a list of work that must be done to deliver value to your customers. This list will continue to evolve over time and is ever-growing. The Scrum Guide is an excellent resource for learning how the Product Backlog fits within the Scrum process. The backlog should always be visible to the team, management, and customers. Backlog items can include big stories that lack refinement (to be completed in the future) as well as detailed user stories (to be completed immediately). Managing the Product Backlog is the prime responsibility of the Product Owner (PO) role.

The Product Backlog should be a simple, prioritized list of work to be done. The backlog list should include the user story title, priority ranking, and story estimate (which describes the amount of effort it will take to complete the story). Whether your agency uses software or physical cards, you'll need to include what your "Definition of Done" is and a statement of customer acceptance. During the Sprint Review, the team and customers identify whether acceptance was met or not.

Why do we need a prioritized backlog?

Gallup performed an employee engagement study on 30 years of its own data that included over 17 million employees. Gallup identified twelve core questions (Q12) that every employee

desires. The top concern of employees based on the study is that they want to know what is expected of them at work. In other words, employees want clear and concise direction.

The Product Backlog provides clear direction because it has been prioritized by management and/or customers. An effective backlog also provides transparency because it is posted where all can see. As priorities change (and they will!), the backlog is the first tool the PO uses to identify where new work belongs.

It's imperative that the Sprint Backlog -- containing items the team is currently working on during a Sprint -- remains undisturbed if at all possible. The Product Backlog can be reprioritized at any time as long as it does not interfere with or include any work items from the Sprint Backlog while the team is in a Sprint. "Feel free to reprioritize Product Backlog items that the team is not working on as often as you wish," says Jim Highsmith, an Agile Manifesto signatory.

Sprint Backlog

The Sprint Backlog should include the highest prioritized items from the Product Backlog. The Sprint Backlog represents what the team has forecasted to complete by Sprint's end. The Sprint Backlog is a subset of the Product Backlog, and is the result of the Sprint Planning meeting.

The Sprint team will usually translate the user stories into development tasks when they move them to the Sprint Backlog. Normally, a task is a work item that can be completed in one day. Simply put, if the user story is "what" needs to be done, then the task is "how" the work gets done. Identifying the tasks that get user stories to "done" is the responsibility of the Development Team only. POs, managers, and customers can help design user stories, but only the Dev Team can build the tasks. "The Sprint Backlog is a highly visible, real-time picture of the work that the Development Team plans to accomplish during the Sprint, and it belongs solely to the Development Team." (Scrum Guide, 2013).

Backlog Refinement

The Backlog Refinement meeting is an opportunity to look ahead to future sprints and identify or refresh the Product Backlog. The meeting is not a formal Scrum ceremony, but it is very useful as your team matures. The refinement meeting is a less formal version of the Sprint Planning ceremony. The Sprint team reviews the Product Backlog and identifies any new or

reprioritized backlog items. The idea is to have enough user stories estimated for two or three Sprints ahead.

Refinement meetings are conducted weekly, typically in the middle of a Sprint – not at the beginning or end. The current Sprint can be discussed, but the meeting is designed to look forward. The team can estimate user stories in the backlog for future Sprints. If a team has enough "points in the oven" -- enough capacity to handle more work -- the Sprint Planning meeting should be a breeze. The Product Backlog is constantly being refreshed if the team meets weekly for the Backlog Refinement meeting.

Tips on Backlog Refinement

- While planning, resist the urge to spread out in scope too quickly. Respecting vertical slices and being careful to decouple Epics tightens the overall scope, and makes the backlog more manageable.
- User story mapping is a good exercise to identify user needs and to keep the backlog from getting stale.
- If requirements shift during the development process, close the original ticket, and create a new ticket that reflects the new story.

Workflow

This exercise should be completed with two or more people. One person should be designated as the Product Owner while the others will be stakeholders. As the group reviews the stories, the Product Owner accepts input from the stakeholders but retains authority to decide on final prioritization. You will be using a storyboard, which is a tool that helps the team keep track of items in a project, moving them along from "Backlog" all the way to "Done".

>> Learn more about storyboards.

For this exercise, you will be reviewing and prioritizing features for a new car. To do this, we highly recommend that you access the free project-tracking software at Trello.com.

>> Learn more about Trello.

Using Trello

You will be using a Trello template that is full of potential features. Assuming you have signed up for Trello and are ready to start the exercise, go to https://trello.com/b/wgxpToaw and copy the board. (To copy the board, go to the Menu on the right side of the screen, click "More" to see the options, and then click "Copy Board". The resulting board is your own to manipulate during this exercise.)

Using Paper and Pens

You can also use a big board with sticky notes for this exercise, but we suggest that you will eventually want to move to a project-tracking software like Trello for managing real-life projects. Write the following user stories onto sticky notes, which you will move from a column labeled "New" into a column labeled "Backlog". The numbers should also be written on the notes -- they represent the "points", or the amount of effort required for the Dev Team to complete the story:

Power steering (3)
Hydraulic brakes (3)
Heated seats (5)
Air conditioned seats (8)
1000-watt speaker system (13)
Fog lights (3)
Sun roof (3)
Push button start (5)
Bluetooth integration (3)
Cup holders (1)
All weather floor mats (1)
Parallel parking assist (8)
Programmed seat position (8)
De-icing windshield (5)
Satellite radio (3)
Air conditioning (5)
Power windows (8)
Power locks (5)
Power side view mirrors (3)
Turn by turn navigation (8)
Bike rack (3)
Rear door child locks (3)
Roof rack (3)

Activities

Prioritize the Backlog

Your first task will be to review all of the stories in the backlog and determine their rank relative to one another. When ranking, you may cluster related stories near one another. However, it is important to ensure you provide a breadth of features as the new car is being built. For example, heated leather seats are great but if there aren't any cup holders, you're bound to spill some coffee.

As you review the stories, drag them from the New column to Backlog, placing them in the priority order of your choosing. Throughout this process, you may reshuffle the order or have a discussion about the relative importance of one feature versus another. This is a vital part of the backlog prioritization process. Once all of the user stories in the New column have been moved over to the Backlog column, you are done with this task!

Scope a Sprint

Now that you have prioritized your backlog, you need to determine what will go into the next sprint. At the top of the In Progress column, you will see the number 34. This is your sprint team's average velocity. Review the stories in your backlog and place 34 points worth of stories into the In Progress column. You may have to place a lower priority story into the sprint so that you can reach 34 points. Do not go over 34 points. You have just planned for an upcoming sprint!

Refine and Scope Next Sprint

Fast forward a few weeks. Pretend your team is about to complete their sprint, and you are looking forward to all those new features for your car. It is time to start planning your next sprint!

If you are using Trello, copy this board https://trello.com/b/mHfceaCf to get started.

If you are using sticky notes, you will need to re-group the stories into the following columns (some new features have been added):

New:

Turbo charger (8)

Fold down rear seat (3)

Wood trim (5)

Rear seat entertainment system (5)

Steering wheel mounted audio controls (5)

USB charging station (5)

Backlog:

Air conditioning (5)

De-icing windshield (5)

Turn by turn navigation (8)

Rear door child locks (3)

Roof rack (3)

Sun roof (3)

Programmed seat position (8)

Satellite radio (3)

Bluetooth integration (3)

Bike rack (3)

Power side view mirrors (3)

1000 watt speaker system (3)

Fog lights (3)

Push button start (5)

Air conditioned seats (8)

In Progress:

Hydraulic brakes (3)

Heated seats (5)

Power windows (8)

Power steering (3)

Parallel parking assist (8)

Power locks (5)

All weather floor mats (1)

Cup holders (1)

Since your initial backlog prioritization meeting, a handful of new features have been suggested by the team and stakeholders. You can even add in some of your own ideas at this point too. Get started by reviewing the stories from the New column and place them into priority order along with the other stories in the Backlog column. As you add the new stories to the Backlog column, feel free to reprioritize the Backlog column as necessary. It has been a few weeks -- the features you and your stakeholders desire may have changed!

You have successfully prioritized your backlog and are ready for the next Sprint Planning meeting!

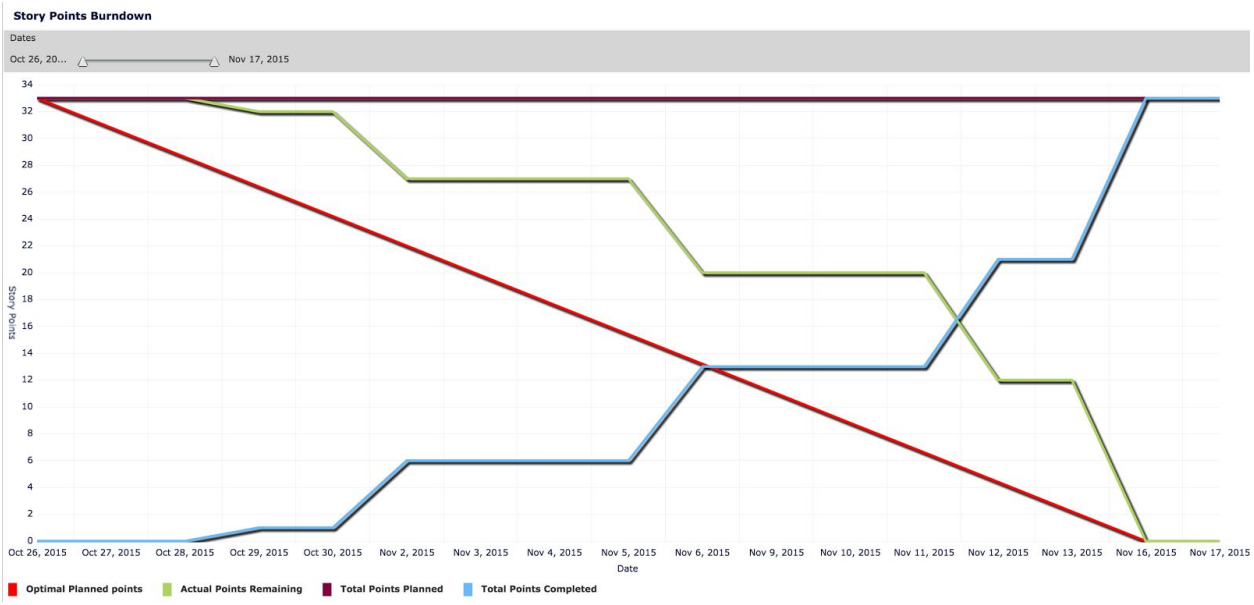
CONGRATULATIONS, YOU'VE COMPLETED LESSON 3!

Lesson 4: Interpreting the Burndown Chart

This lesson provides insight into the various metrics tracked by a burndown chart and the value that each delivers.

The Burndown Chart

The progression of stories and their assigned points, as they reach a completed state, can be expressed in a burndown chart. A burndown chart does not count partial progress -- only stories that are complete may count towards the "burn". As a result, the Product Owner and the team can identify how much of the product can be delivered at any one time. A burndown chart for the Sprint should be a regular part of Scrum meetings as it helps the entire team identify whether or not they are on track. The progression of stories and their assigned points (representing the effort required to complete them), as they move incrementally from Backlog to Done, can be expressed in a burndown chart, shown here:



Burndown charts, like storyboards, can vary in appearance and have a number of tracked values, but they all have these things in common:

- Red line - Expresses a vector from the maximum anticipated story points to zero -- the ideal “red line”. This vector is the perfect function of points completed or “burned” per day.
- Green line - Shows a plot of of actual points complete for each day that passes. If you finish a 5 point story, 5 points are burned. The actual points completed may fall above or below the ideal schedule (red line) as the Sprint progresses. This is to be expected.
- Purple line - Shows the number of points planned. This number can go up or down if the team adds a “stretch story” or a story is removed because it’s blocked by external circumstances. In the example above, the team does not add or remove any stories to the Sprint, therefore the line stays flat.
- Blue line - Sometimes (as pictured above) there is a burn-up line, useful for predicting team patterns to find the “break even” point.

Every team has a maximum point value that they can take on. New teams tend to underperform and experienced teams tend to burn hot. It takes a few Sprints to dial in on the team’s point maximum.

A burndown chart can provide value to the PO and the team, both during a Sprint and after it has been completed. While a Sprint is ongoing, you can review the burndown chart for indicators that help you infer how well the team is doing or if there is a potential issue on the horizon. After the Sprint, the burndown chart provides information about how the work was completed. Below are three burndown charts, along with explanations that describe key takeaways during and after the Sprint.

Example 1: Sprint with Large Stories

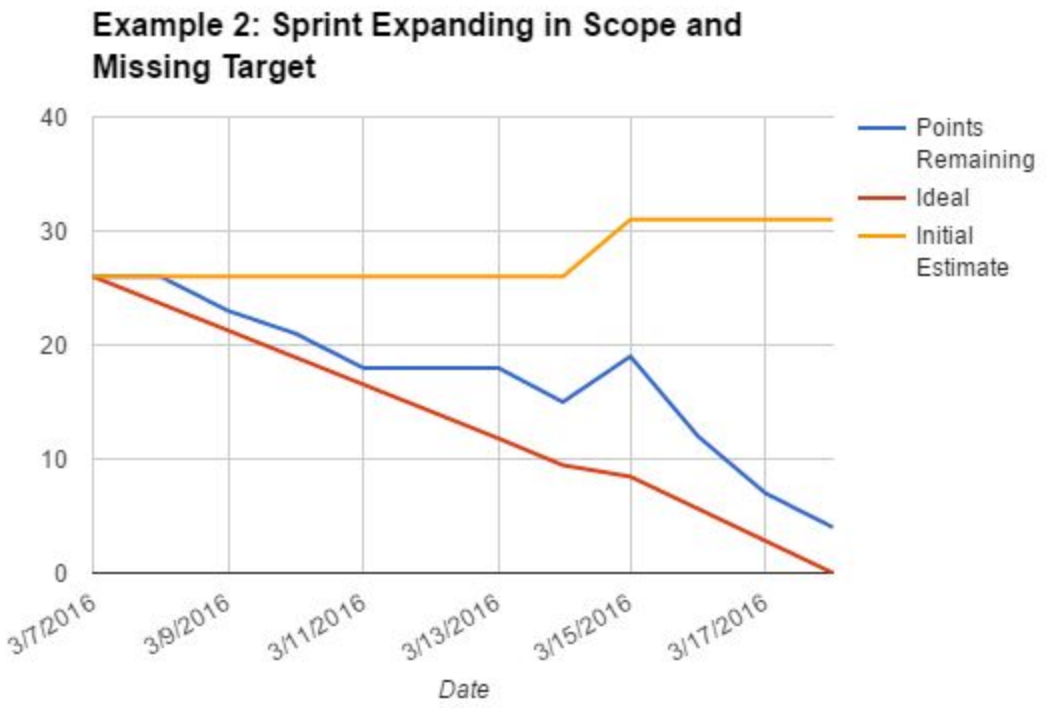
During the Sprint

Throughout the Sprint, viewing a burndown chart like the one in Example 1 can highlight stories that are very large and complex. For instance, on March 11, one user story may have been completed but it accounted for almost a third of the total points in the sprint. Stories that make up over a quarter of the points in a given Sprint can be considered risky. The larger the story, the greater the amount of uncertainty (i.e., possibility that the work will not be completed by the end of the Sprint).

After the Sprint

Example 1 may also indicate that a team is still functioning in a waterfall manner. For example, 4 stories totalling 14 points may have been completed on March 11, and 5 stories totalling 16 points were completed on the 14th. This may indicate that the team is not dividing their work properly or that there is a bottleneck in the process.

Example 2: Sprint Expanding in Scope and Missing Target



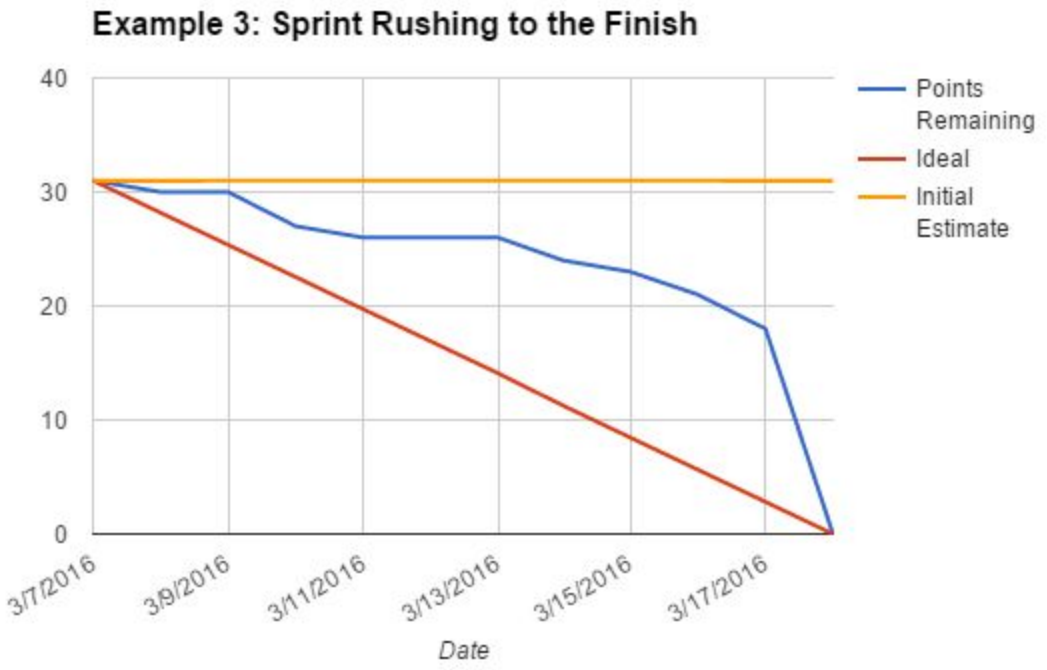
During the Sprint

In the first week of the Sprint, the team in Example 2 seems to be on pace. However, on March 14, additional stories were added to the Sprint, as demonstrated by the upward turn of the yellow line. This places the team in a precarious situation where they may not be able to complete all of the work required.

After the Sprint

The effect of adding stories is fully realized when the team does not deliver all functionality on the last day of the Sprint. Adding scope (stories) to a Sprint after it has started is considered a bad practice. Many times, a PO will be tempted to add stories mid-Sprint if a requirement was missed or it is discovered that a product cannot be shipped without an additional feature. This becomes quite risky and can result in the team missing their original target as shown above (or results in the team working long hours to make up the difference). When a team is working long hours to catch up, they are more prone to shipping a defective product. This will result in downstream maintenance costs that could have been otherwise avoided.

Example 3: Sprint Rushing to the Finish



During the Sprint

Throughout the Sprint, you will notice that that team lags further and further behind the expected completion schedule as denoted by the red line. This may be due to the fact that the team is blocked either externally or internally. An external blocker, for example, could be that they are relying on you for the language that goes into an automated e-mail message or they need the networking team to make a change that is out of their control. As a Product Owner, one of your chief responsibilities is to unblock your teams as quickly as possible. Sometimes that means calling the network team on behalf of your Development Team or stopping your day-to-day work to complete the language for an automated e-mail. Internal blockers can arise if the team has not built that particular type of functionality before, or if one of their resources is busy on other tasks.

After the Sprint

Example 3 demonstrates a team who lagged behind for a majority of the sprint and suddenly completed the remaining stories on the last day of the sprint. While they did finish on time, the fact that 18 of the 31 points (almost 60%) were completed in one day raises a number of questions. Were the stories so interrelated that they could not test until the end? If so, that is an indicator that the team could have broken down the stories to be more independent from one another. Was there a bottleneck in the development process that forced the team to rush to the finish?

As you learn to interpret and create burndown charts (it takes practice!) you will be more effective at helping your team achieve their goals on time.

CONGRATULATIONS, YOU'VE COMPLETED LESSON 4!

The Next Level

There are many important topics which have not been covered in this introductory study track:

- Automated testing
- Continuous integration
- Modern deployment tools
- Lean startup continuous learning techniques

And many more! Once your agency begins to reap the benefits of Agile, you'll likely want to continue studying and pursuing this methodology that allows governments to bring delightful, effective services to citizens and customers.

You're done!

You've completed Agile for the Government Product Owner!