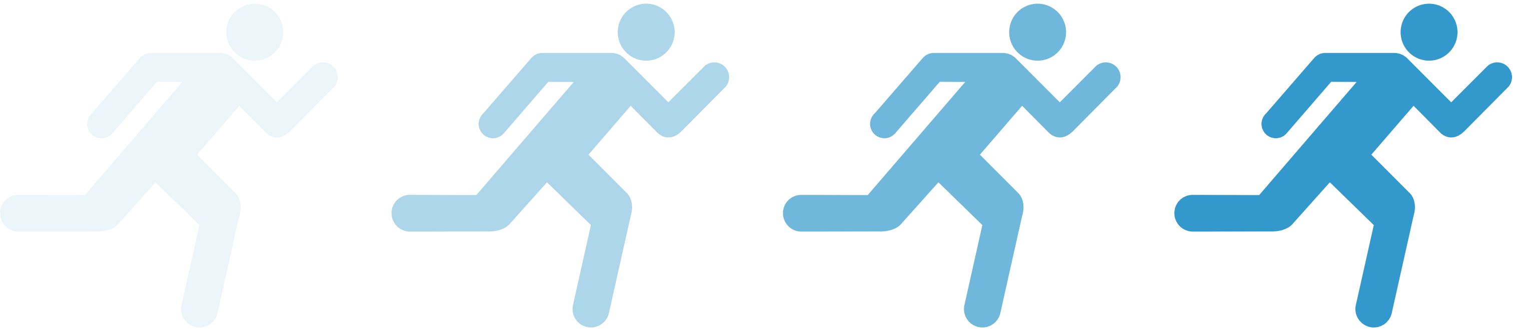
The Lean-Agile eBook Practitioner Series

FIELD GUIDE TO AGILITY

A Complete Illustrated Guide For Pragmatically Implementing Agile Software Development



AXA redefining / standards®

FOREWORD

Welcome to AXA's Field Guide to Agility!

In today's marketplace, agility is no longer optional. All companies must learn to quickly adapt to ever-changing customer demands, or they will fall quickly behind their competitors. While there is no miracle cure, more and more companies are finding that organizational agility, and the supporting processes that enable teams to more rapidly adapt and deliver new products and services, are key to competing in a rapidly evolving marketplace.

But how exactly do you implement agile practices in an organization?

Fortunately, this is where this Field Guide to Agility comes in! Within its digital pages, we will use simple, straightforward language to describe what you need to implement a mature agile software and product development process, including:

- Key foundational principles
- Best practices
- Streamlined methods
- Advanced techniques

Our goal is to provide a springboard for all stakeholders involved in conceiving, building, and deploying new software products and services. We have carefully constructed it at a summary level to make it useful not only to developers, but also to executives, management, and others who have critical roles within the development life-cycle. Our guide presents all facets of agile from beginning to end, with particular emphasis on:

- Product planning
- Management
- Execution
- Collaboration and software release
- Key Performance Indicators

This guide broadly covers the most common and proven agile methods and practices.

The concept of Agile is relatively simple, but mastering it requires time, effort, and dedication. If you are serious, this will become a lifelong pursuit. As you will soon learn, committing to agile is a journey of continuous improvement.

CONTENTS

1. AGILE VISIONING & LONG-TERM RELEASE PLANNING

"Without Vision, the People Will Perish"

FROM IDEA TO PRODUCT LAUNCH

"Imagination is the beginning of creation.

You imagine what you desire, you will what you imagine, and at last you create what you will."

~ George Bernard Shaw

The goal of any software development life cycle is to get a product from conceptual phase to launch phase, from idea to working software. Along the way, however, it is necessary to deal with many unexpected developments that makes keeping the process moving smoothly an ongoing challenge. The agile system described in this book, Scrum, helps to smooth out the rough patches you will encounter along the way.

In this field guide, we focus on a pragmatic application of the Scrum project management framework, an agile model for executing software and product development projects. Before we get too far into the details of Scrum, though, we need to spend some time discussing the big picture.

Key Terminology

A

Agile
A group of software development methods (Kanban and Scrum are two examples) that share several common principles, such as using an iterative approach and a product backlog to organize work.

S

Scrum
The name of a specific type of agile process . . . the one that we are focusing on in this Field Guide. The name originates from a rugby term referring to the way a game restarts after a minor infraction.

PB

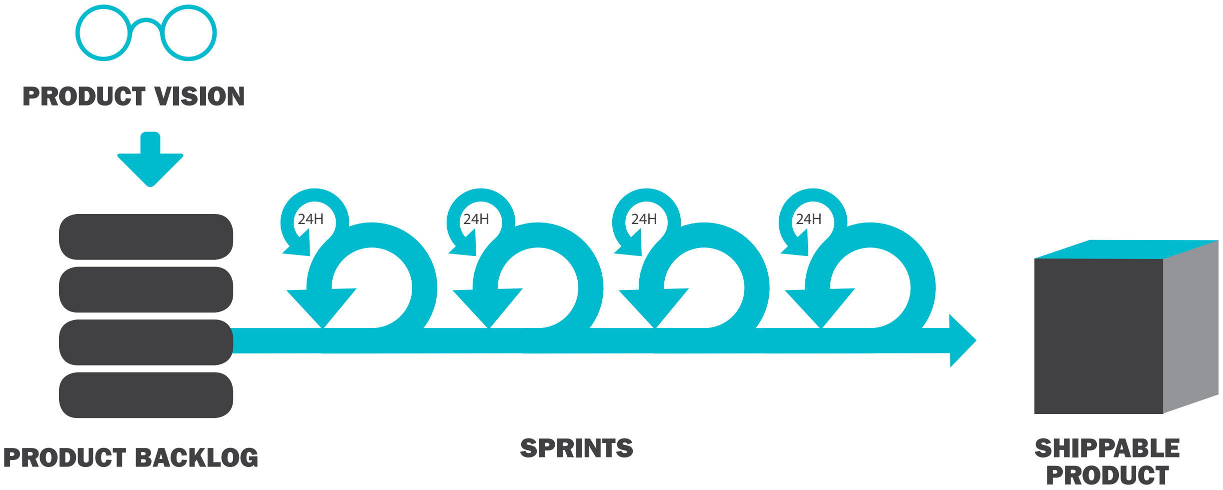
Product Backlog
This is a comprehensively organized list of prioritized features, including customer requests by priority. The Product Backlog is one foundation of an agile system.

Sp

Sprint
A name for each work iteration cycle in Scrum. Basically, a sprint is a bite-sized chunk of work that the team is developing as a discrete unit.

THE BIG PICTURE: PRODUCT VISION

Everything in development needs to be based around a central, well-articulated Product Vision. The Product Backlog (this is the Scrum method of organizing and prioritizing software requirements, which will be explored in more detail shortly) must work together with the Product Vision to create a series of "Sprints" (again, a term we will detail later on) to reach a Shippable Product, in the following process:



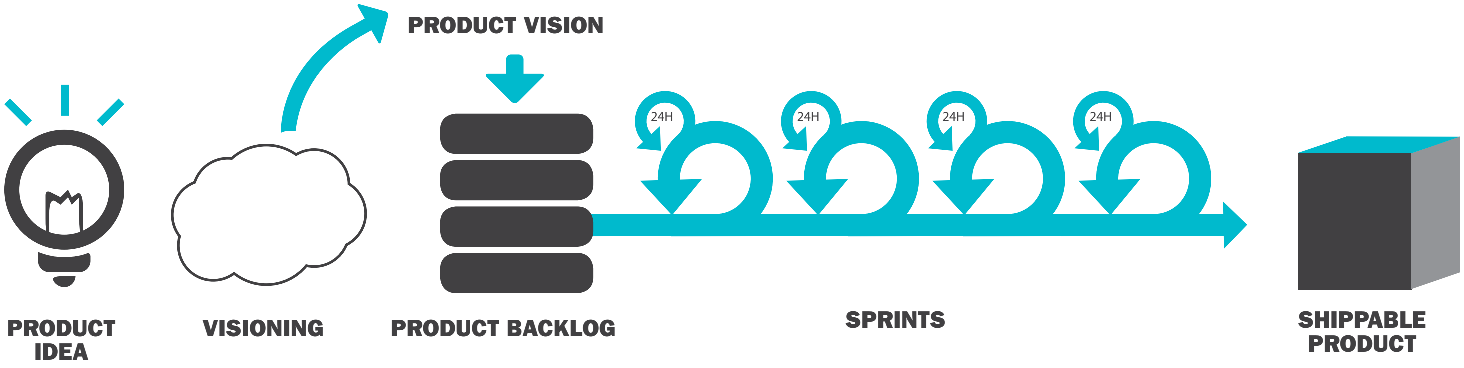
From Product Vision to Shippable Product

The diagram featured above, however, is only partial, because it takes into account only the "development" phase of a product innovation sequence. The front end seems to be missing, so from where does the Product Vision and the Product Backlog originate?

Scrum does not— in fact it cannot—give you a Product Vision. This is not its job. Scrum's job is to organize and prioritize work to meet the Product Vision, and you must develop your Product Vision before actually implementing Scrum.

ENVISIONING YOUR PRODUCT

The process can be extended in terms of the illustration below if we accept the simple ideal that any innovation originates with either an idea to create a new product or refurbish and update a preexisting concept:



From Product Idea to Shippable Product

The figure above illustrates a two-step process:

1. The "visioning" step during which the product vision and the initial product backlog are created

2. The vision is then turned into a product ready for launch during the "development" step (the part that Scrum helps with)

Visioning should get you rough answers for the following questions:

- What will the future product or version look like? What will it do?
- Who are the target customers and users?
- How does the product add value?
- Why is it beneficial for the organization to develop the product?
- Is the product feasible? How will it be built?
- What is the target launch window?
- What is the target budget?

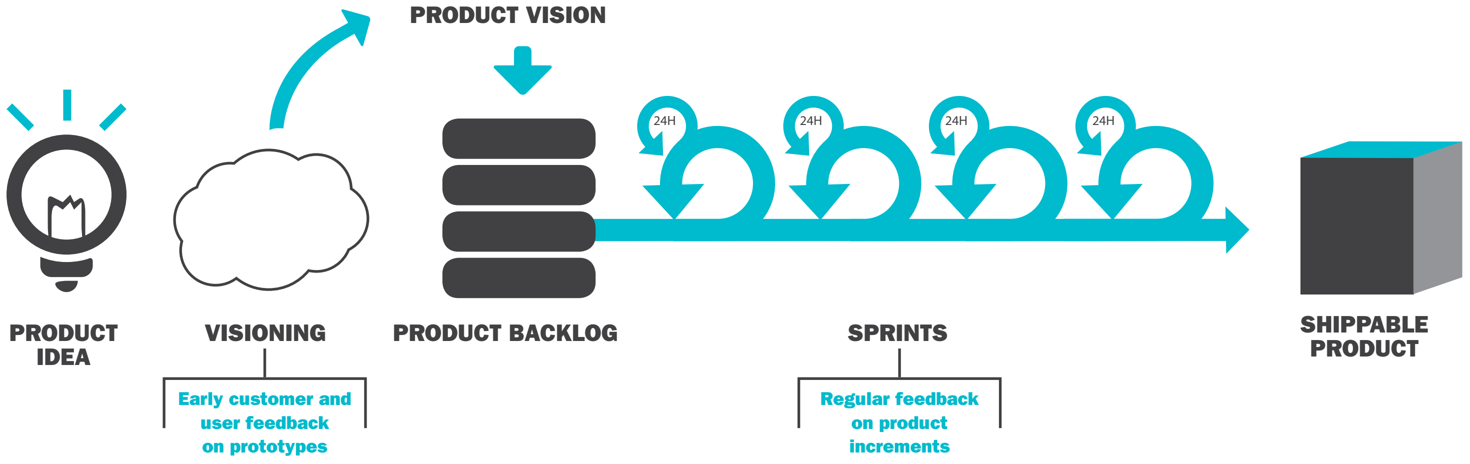
To answer these questions, you may be required to create additional artifacts, including an architectural vision, prototypes, and a business case.

As with all agile processes, envisioning the product ought to be a collaboration, but the Product Owner, a key role in Scrum, should be responsible for facilitating the visioning work by introducing key input from the right people, including:

- Team members
- The Scrum Master
- Target customers and users
- Other stakeholders

This approach brings together the collective knowledge and experience of each individual with the result that any vision is genuinely shared. Retain the same individuals involved throughout the process to avoid handoffs, failures, waiting, stoppages, and other waste. (Note that it is not necessary for all team members to take part in the visioning.) Use prototypes, or better still, product increments to validate prototypes. Avoid airy predictions. Instead, allow the product to evolve progressively by maintaining continuous dialogue with users, customers, and the other stakeholders.

ENVISIONING YOUR PRODUCT



Early and Regular Feedback

It is a mistake to disregard or overdo the visioning work. Wasting months of time in detailed up-front market research, product planning, and business analysis is undesirable in an agile environment characterised neither by stability nor predictability.

The two factors listed below make it easier for you to accurately gauge the weight of visioning effort likely to be necessary for your product:

- The product's life cycle stage
- The product's complexity: the feature set as well as the technical solution

The younger and more complex a product, the higher the necessary up-front investment tends to be.

> As you envision your product, it is important to keep in mind a minimal viable product or that product meeting the selected customer's most basic requirements. Both the visioning effort and the time taken for a product to reach market can then be reduced. As a rule of thumb, it is always best to consider a span of hours and weeks rather than weeks and months.

One way to achieve a reduction in visioning work is to focus on the next product version and then envision the least functionality necessary to addresses a narrow set of customer needs. Thereafter, quickly release a preliminary product or a demo and monitor feedback to determine if you are anywhere near the target. Then adapt. If you would like more details and suggestions regarding the visioning process, the next couple of sections will provide some additional insight. If you would like to move ahead to learning about the Product Backlog, then move directly to the chapter "Crafting Product Backlogs for Agile Success."

HOW MUCH VISIONING IS NECESSARY?

Vision without action is a daydream. Action with without vision is a nightmare. (Japanese proverb)

A common failure of teams and companies new to Agile is to either over or under invest in product visioning. Activity during this phase is always crucially important to keeping teams focused on both determining and creating the correct value for end users/customers.

$E=MC^2$

What is the optimum formula? Actually, no precise formula exists, but the following two key indicators point to the degree of time and energy output likely in visioning a product: a product's life cycle stage and its complexity.

As a general rule, the more immature a product, the more visioning is typically required. Several weeks of visioning in the form of prototyping will almost always be necessary to accurately assess product design and architectural variables. Compare this to the evolutionary improvement of an older product, which generally requires minimum visioning. This is also true regarding complexity. A more intricate product will naturally absorb more visioning effort in terms of developing both the architectural and technology features of a product, as well as its functionality.

When determining your visioning effort, two common mistakes should be avoided:

- Understanding as much as possible about what the future product will both look like and do before plunging into the first sprint
- Do not overinvest in visioning work

It is impossible to guarantee that any visioning is accurate, since no market research technique can deliver forecasts that are 100 percent accurate. Once again, however, we recommend that you limit visioning time and effort to only what is necessary. Try a few of the following ideas to find that optimal balance:

1. Concentrate on about three to five key features of the products that may satisfy customer needs.
2. Envision a product with the minimum of functionality that still offers a tangible value proposition.
3. Get your product prototypes into the market as quickly as possible in order to accumulate customer and user feedback on preliminary product increments as a means to validate and refine the vision.
4. Limit complexity—a simple product is always easier to use, extend, and maintain than a complex version.

2. CRAFTING PRODUCT BACKLOGS FOR AGILE SUCCESS

“A good plan is like a road map: it shows the final destination and usually the best way to get there.” ~ H. Stanley Judd

WHAT IS A PRODUCT BACKLOG?

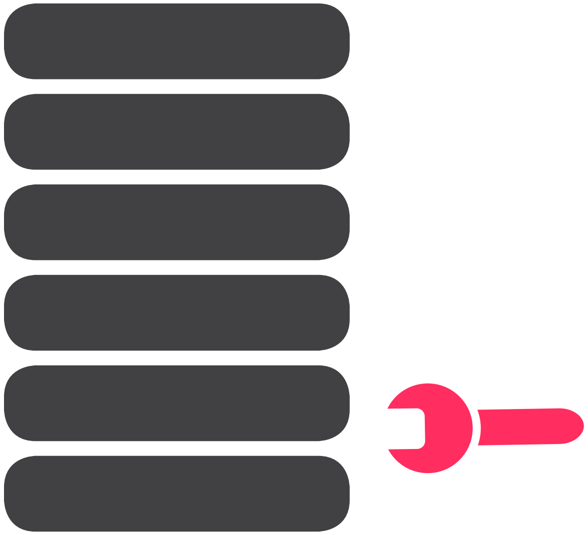
A product backlog is a comprehensively organized list of prioritized features, including customer requests. It is the foundation of the agile system, because it ensures that you deliver exactly what your customer needs.

An accurate and up-to-date product backlog is your guarantee that the team is always focused on the key products and features for your business.

Customer needs and wants lie at the very core of any product. In this regard, Agile practices are effective for one simple reason: they deliver value to the customer in observable increments, step-by-step, giving customers clear opportunities to offer relevant feedback at various milestones and waypoints in the development cycle.

Maintaining your backlog is just common sense, since an outdated backlog will likely cause your team to waste time on features that no longer offer the best value for the business. Even worse, should the product backlog not offer enough features to keep the development team working, then they will remain idle until new features are added!

Fortunately, the agile approach can prevent both of these possible scenarios.

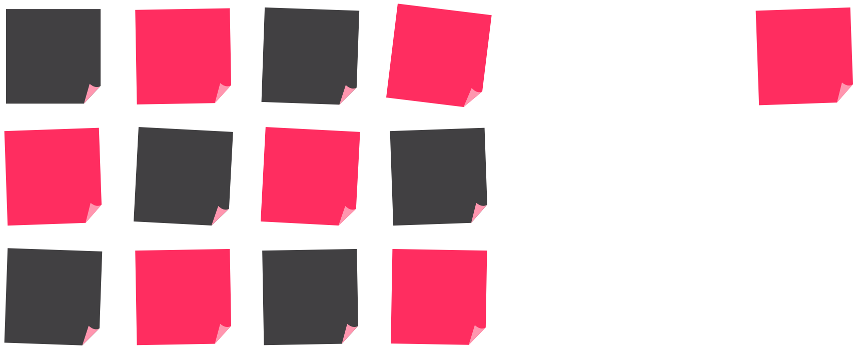


LET'S GET STARTED!

There will come a time when you believe everything is finished. That will be the beginning.

~ Louis L'Amour

Before any practical work can begin, even before your team starts delivering features, the Product Owner must first construct a product backlog for the project or product.



If you are beginning a project from scratch, this process will be relatively clear-cut. The Product Owner will meet with the customer and end users to discuss which critical issues this specific project will aim to overcome with this product.

Though this may seem simple, do not try to rush this process. It is easy to think you can quickly zip through the key customer demands and quickly set up a backlog, but important things could be overlooked.

Taking time up front to plan and prepare properly can make a product even more valuable to the customer.

You should be absolutely thorough during the discovery process. Depending on the project, anywhere from a few hours to a few weeks may be needed to outline enough essential features for the first few sprints, but this will be time well spent. Among other things, this could save the development team from later wasting time in pursuit of dead-end solutions.

If you are working on an existing product, then constructing your preliminary backlog will not be quite as straightforward.

EXAMPLE: Consider a Product Owner who is faced with creating a product backlog for a large legacy application. Depending on the size of the application, perhaps thousands of feature requests, bugs, and miscellaneous tasks have been cataloged over the years.

The problem now is not having too little to do but determining which items are the most important for the team to engage on next. This task may seem daunting at first, but your experience on this project does give some insights that can help. You should have a general idea of which features are higher priority, so initially these can go at the top of the backlog.

No matter where you start, as the project advances, the Product Owner is responsible for continually pruning and prioritizing the product backlog, with input and continuous feedback from customers.

ADDING FEATURES TO THE PRODUCT BACKLOG:

WHAT GOES IN THERE?

Bugs

Bugs and defects are problems that have been found during development and testing, as well as by end users. In “waterfall” processes, testing is typically the final step in the development life cycle. As a result, code is often released that includes defects, which means that bugs pile up over the years. These need to be included in your backlog and prioritized accordingly.

One of the benefits of agile is that testing takes place throughout the process, dramatically reducing the propagation of defects. Although they will still occasionally show up, this is the agile method, not the miracle method.

New Features

Feature requests will originate from a range of sources, including end users, sales, support, and product management. Because of all the varying conditions that must be satisfied, the following issues can often be the most difficult to prioritize:

- The existing customer base
- Needs of near-term sales opportunities
- Working toward a longer-term vision of your product

How do you get new customers without alienating your core of trusted clients?

The buck stops with the Product Owner. They will need to routinely monitor all these sources and have the last say when it comes to potentially conflicting requests to ensure that the backlog contains features that will attract new customers but still build and retain loyalty among existing customers.

Technical Debt

As a project progresses, a product’s scope and direction will probably change as all the details begin to emerge and the finished product begins to take shape. Technical debt refers to backlog items that accumulate due to new conditions. The name comes from the idea that they frequently accumulate over time. To avoid becoming lost in the planning cycle, these tasks should certainly be included in the product backlog and prioritized alongside defects.

Performance and scalability expectations will change. New technology or unforeseen better practices will become available. Everyone can agree that it seems like a good idea to update the existing solution to address these issues. In practice, though, it can be difficult for the Product Owner to prioritize these new conditions over more obvious features that have been explicitly requested by customers.

Recall the danger of having a product backlog that does not contain enough work to keep the team busy. An agile tool to prevent such a development is to carefully track technical debt. As changes are made, the Product Owner will continuously add technical debt tasks to the backlog.

Examples of technical debt include upgrading to the latest third-party libraries or improvements to the code itself, e.g., making architectural alterations for improved scalability and configurability or refactoring the source code for improved future maintainability.

ORGANIZING THE PRODUCT BACKLOG:

KEEPING NEAT WITH CLASSIFICATION AND PRIORITIZATION

A place for everything, and everything in its place.

~ Isabella Mary Beeton

The Product Owner not only creates product backlog but must also classify and prioritize the tasks on it. Anyone can see that a backlog is essentially pointless if you can't analyze it quickly by different criteria. Although classification and prioritization are primarily manual procedures, the process usually takes less time than generally imagined.

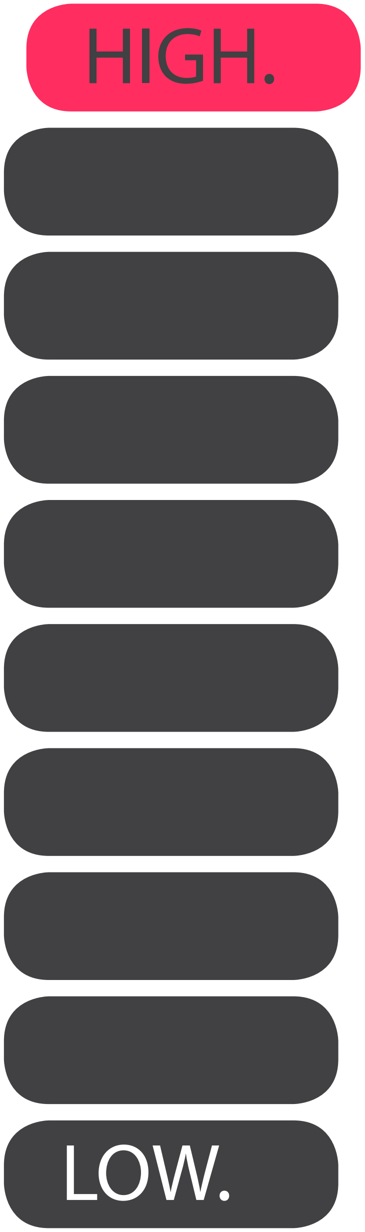
Do not focus too much on organizing your whole backlog when you first get started. Instead, locate and mark the features that are quite obviously high priority.

Be sure also to call for the development team to classify priorities themselves, as indicated below. A gradual approach is best, and you will no doubt begin to progressively apply prioritization and classification to the backlog at a point when each feature or issue needs to be addressed, based chiefly on your anticipated sprint schedule.

Classification

We suggest getting started with the following two classification categories:

1. Functional Area: By tagging items based on the functional area they affect, you can coordinate your work to address several issues within that specific functional area at once. Developers can help classify these work items because, as they are involved in the basics of the underlying code of the project, they will already have an in-depth understanding. Don't be afraid to use this expertise! For example, if you are focused on one area of top priority (say, refactoring the user security model), then planning a sprint to address that particular area of the product is going to be easier if your backlog highlights it specifically, which is the result of tagging by functional area.
2. Theme: Alternatively, classifying according to theme is primarily a process of grouping features together to achieve a smoother process of sprint planning. When planning a sprint, marking each feature with a theme offers greater plasticity over what you ultimately hope to achieve. Let us imagine that your customers are having trouble finding data in the application you are developing. Tagging your backlog with individual theme tags—one example is usability—makes it much easier to build a sprint to address immediate customer concerns. In this instance, you simply have to pull out everything with a usability tag, using the associated features to map out your sprint.



Low-priority items will always seem to be at the bottom of the pile. This does not mean, however, that they have vanished; so do not get tripped up by complacency! To avoid lurking, low priorities causing a nasty surprise later down the line, classification by theme helps by making it much easier to find related items during planning, no matter their priority.

Prioritization

As a next step, the Product Owner is required to prioritize features in the product backlog by business value.

It is here that the Product Owner's grasp of the requirements of customers and end users is critical. Just because you can't put a specific price on something doesn't mean that it isn't valuable, and this applies to business value prioritization as much as anything. Even though a feature value may not be counted in cash, features in the product backlog need a clear business value to be considered for development.

A clear understanding of the Product Vision will help guide the Product Owner during this phase, which is why the first chapter focuses on this so heavily.

An optimal product backlog is one with the highest-priority feature at the top and the lowest at the bottom. It may, however, particularly when dealing with large feature sets, make more sense to use a bucket approach. In this case, descriptions such as low, medium, or high help to rank and group features with similar priorities. You can then refine those buckets as much as you like after features are grouped into their own buckets.

What must be remembered is that before the development team begins working on the highest-value features, it is vital for an organized backlog to be in place with a good classification and prioritization scheme.

At the same time, you want to avoid the analysis paralysis that is the bane of waterfall methods, where you spend too much time focused on analysis. Keep the level of detail in the backlog at the 10,000-foot level.

Agile ensures that the customer's needs are at the core of the product right to the end of every sprint. One must always make sure not to get sidetracked and prioritize features according to the "loudest customer" (assigning oil to "squeaky wheels" so to speak) or fads such as "this week's sales opportunity."

As is always the case in business, there is not much room at the top: if you add something to the high-priority bucket, you'll have to take out something else. If you use the bucket approach and keep yourself disciplined, e.g., making sure there are no more than five features in high priority, you will ensure there are never more items in the high-priority bucket than you can handle.

MAINTAINING THE PRODUCT BACKLOG:

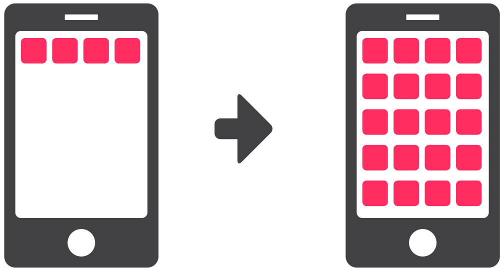
KEEPING UP

A good day's work is never done, and likewise the work of building and organizing the product backlog never stops. If you follow agile methods, however, this will not necessarily be a chore, as you make sure that what you are working on is exactly what the customer really needs. The Product Owner must commit to continually updating the backlog on a regular basis for the following reasons:

First, customer needs are dynamic, and the backlog must keep up with those changes. You have to make sure your product is not dated as soon as it arrives.

EXAMPLE: **Flip-up mobile phones were all the rage a few years back, then mobile phones with QWERTY keyboards, while now it is touch screen smart phones that dominate the market. If we were to make a product backlog for the input device of a mobile phone, it would be very different these days compared with a couple of years ago.**

Staying abreast of those trends and adjusting priorities accordingly is the Product Owner's responsibility. In this way, implementing features can also change or even remove the need for other seemingly unrelated features.



EXAMPLE: **If a credit card processing feature has been added to the product backlog behind a second feature integrating the PayPal payment system that has already been proven to be more effective, then it is likely that the credit card processing feature will be removed from the product backlog.**

Once again it is the responsibility of the Product Owner to manage such changes, and this can be so much easier by more effective categorization.

Finally, the product backlog is viewable by everyone, but it is up to the Product Owner to compare new ideas with existing ones and determine whether or not they should be prioritized into the backlog. As we said before, the buck really does stop with them.

POPULATING THE SPRINT BACKLOG:

WHAT TO TACKLE, WHEN TO TACKLE IT

The sprint backlog comprises features that the team itself has selected from the product backlog for a specific sprint. Thereafter, the team makes a commitment to deliver within each current sprint every feature listed in the sprint backlog.

Be sure not to bite off more than you can chew. In early sprints, taking on less work than a team thinks they can handle will actually help establish quick wins and build confidence for teams new to Scrum or other agile methods.

Like any relationship, both trust and commitment stand supreme. The development team must understand the level of their commitment. Unlike waterfall, where a wide range of desired features and good ideas are often removed to meet deadlines (leading to lack of trust from the customer), agile teams value the commitments they make in a sprint in order to build trust. This point cannot be emphasized enough: with agile, the customer is always at the center of product development, eliminating any potential disappointments.

Again, each feature is broken into tasks by the development team once it is known what features will be applied to the sprint backlog.

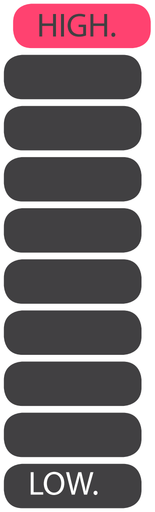
The duration of these tasks should not exceed 10 hours, or two man-days. In addition, team members should not exceed 8 hours per day of committed work. We recommend starting with 5 and then adjusting once a cadence is established.

This process helps the team understand the problem better and get a handle on exactly how much work will be involved. Team members should not assign or grab a task before it becomes a work in progress; otherwise, they will find themselves tripping each other up trying to address too many different issues at once. In order to leave tasks open for other team members to grab when they become available, task ownership is not predetermined.

ROUNDING UP: BACKLOGS IN A NUTSHELL

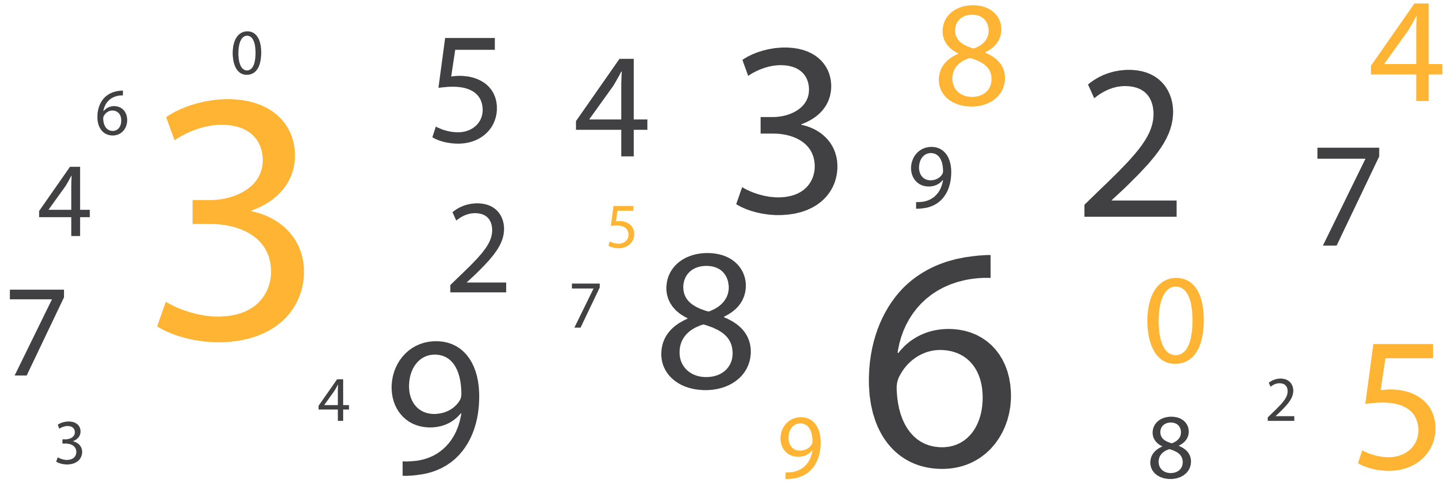
Proper planning and organization are critical to delivering exactly what your customers need, and that is where product and sprint backlogs are important. Here is what we have learned:

- The Product Owner is in charge of the product backlog, prioritizing customer requests, and ensuring that the team is active on the key features.
- The product backlog is a list that can include bugs, technical debt, and new feature requests, which can then be classified in categories labeled as theme and functional area.
- Features in the product backlog should then be prioritized by the Product Owner, usually based on their business value, often ranking them to determine which is tackled first.
- It is important to maintain and frequently adjust the product backlog to ensure that the correct product features are always delivered.
- Features from the product backlog are chosen by the team, and a commitment is made to deliver these in the current sprint. The selected features are broken down into their related tasks during sprint planning.



3. AGILE ESTIMATING: FINALLY, ESTIMATING THAT MERE MORTALS CAN MASTER

ESTIMATION



More than just simple guesswork, estimating can seem like a tedious task to even the most confident and experienced project teams. Your client wants to know how long a project is going to take, but you cannot afford to give any certainties for risk of bonding your team to a deadline they may not be able to meet, and thus jeopardizing customer trust.

However, if your project is agile, then you can breathe more easily, because with agile projects you have nothing to fear but the fear of estimation itself.

When you first start out, your estimating skills certainly will not be much to boast about. In fact that is an understatement.

You will probably be lousy, but don't worry. Estimating is not a precise science, and it makes sense to think of it more as an art. Like all art, estimating improves with practice. Of course, team estimates will not start to match up over night. It will be a gradual process.

As you come over to the agile way of doing things, your initial estimates will be imprecise, so don't expect perfection overnight. Even if you are already experienced at estimation in other development methods, you will not grasp everything immediately.

It will probably take a few attempts over several weeks (or even months) for your team to become proficient at agile estimating. It is vital that you discuss this teething process with your project team, as well as management. Fortunately, agile allows you to more accurately redress your estimates during the project itself.

WHEN SHOULD ESTIMATING HAPPEN?

It’s tough to make predictions, especially about the future.

~ Yogi Berra

Agile estimations are revised continuously. At the start of each sprint, estimations are made specifically for features and then again later for individual tasks.

For Scrum, the very first day of the sprint is devoted to two-part sprint planning. The two parts of this planning (which are often split apart in other agile strategies) are covered in more detail in the next chapter but can be summed up as follows:

- Release planning: The team (coordinated by the Product Owner) decides which features will be part of the next scheduled release.
- Iteration planning (sometimes called sprint planning): The process of splitting up the features for release into discrete development tasks, each no more than 16 hours in duration.

Who Gets Involved With Estimating?

Throughout release planning, the team examines the potential workload of each feature in the product backlog. Members of the core development team are all involved in sizing attempts. Many people are alongside the Scrum Master and the Product Owner or customer, including the developers, testers, and tech writers.

While feature clarification and facilitation is the responsibility of the Product Owner and Scrum Master, estimates are the sole domain of the development team because only the team can be held accountable for concluding the work. This is where agile differs from more established methods, wherein team leads or development managers are expected to supply estimates.

The idea that the customer should be part of this process might amaze you, but consider this: if you involve the customer representative—or Product Owner—instead of fumbling in the dark to calculate the value that a feature should provide, you can simply ask. The Product Owner is able to make estimations a lot more straightforward.

How Long Should it Take to Estimate?

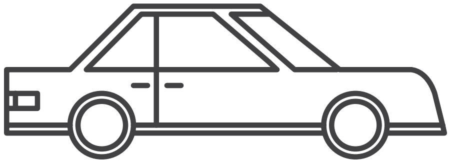
This depends on two main factors: how familiar team members are with one another and how well they understand the sizing techniques in use. Initially, hours may be needed just to size a few features, but once a team has gathered more experience less time will be required. In due course, teams will need only a few minutes to estimate each new feature. If, however, your team still has difficulty, even after a few sprints, then this issue will need to be addressed in the sprint retrospective. An agile coach could also be included.

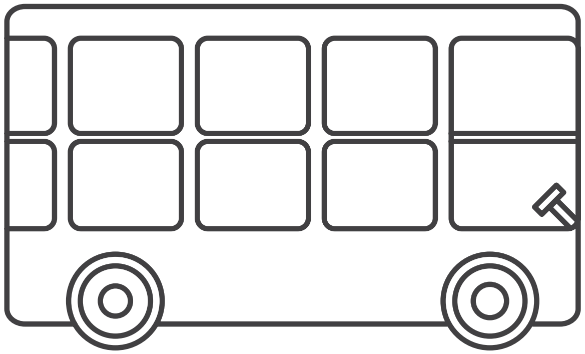
THE POINT SYSTEM

Points determine the relative sizing of features, another departure from traditional methods, in which hours are the typical measurement.

You can use points to size any type of feature or even any requirement artifact, such as a use case or user story. Although point values frequently vary, the Fibonacci sequence is a particularly common means of determining value range: succeeding numbers in the series are generated by adding the preceding two numbers.

2, 3, 5, 8



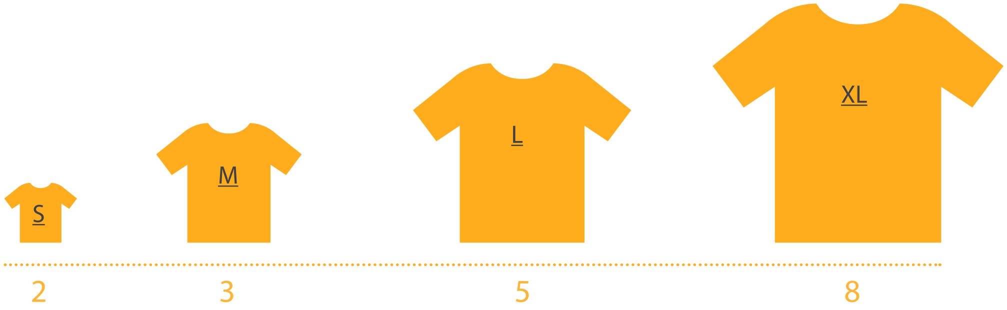


SIZING EXAMPLE: Using the Fibonacci sequence, numbers two, three, five, and eight, presume that we give a car a value of two. In comparing the size of two vehicles, we might also decide that a bus has a value of five, making it more than twice the size of the car. This means that it is bigger than three but not more than four times the size of the car, but smaller than eight.

The Fibonacci sequence is used because it does not just allow us to sequence features by size, which could be done just with an ordinal numbering system like 1, 2, 3, etc., but it also offers a ready way to scale up the points for larger features. An ordinal system will not typically be sufficient, because Points are used to measure team velocity. Velocity can then be used to determine when a certain amount of work currently in the project can be completed. If, for example, forty Points are found in the project and a velocity of five Points per sprint average is revealed by the team, common sense dictates that the team will most probably finish their work in eight sprints, if we assume no other variable and all else is equal.

THE POINT SYSTEM

EXAMPLE: Consider four features of different sizes. Feature 1 is the smallest. Feature 2 is twice as big. Feature 3 is three times as big as Feature 1. Feature 4 is four times as big as Feature 1.



In a regular ordinal number sequence, these would have values of 1, 2, 3, and 4 Points, respectively. In the Fibonacci sequence, however, these same features are classified as worth 2, 3, 5, and 8 Points, respectively.

When the team goes to pull tasks for the sprint, these different Point values matter. A team working in the ordinal system will find that Feature 1 and Feature 3 are together worth as many Points as Feature 4. In the Fibonacci sequence, however, Feature 1 and Feature 3 together come out to only 7 Points. The Fibonacci sequence method includes an inherent compensation that large features like Feature 4 are fundamentally more complex, running the risk of additional problems and delays, and therefore makes them worth more Points than just combining the two smaller tasks.

Eventually, your team will innately sense the relative sizes of things. As we've already mentioned, when a team first transfers to agile methods, the initial estimates between team members will not just effortlessly coincide. Don't panic! This is perfectly normal. You will get better at this with time.
If you size product backlog using Points, you must remember that Points are relative in that they relate to the team's particular skills, so you cannot directly compare the performances of two teams by Point value. This means that if Team A completes 40 Points worth of product backlog and Team B completes 20 Points, it does not necessarily follow that Team A has done more work than Team B.

The numeric system is not the only one available. Although the advantage of Points is that they can be used to mathematically determine velocity, a qualitative measure, for example, T-shirt sizes, are sometimes a preferred option. Using this sizing technique teams size user stories as either small, medium, or large.

USER STORIES

Requirements for an agile project can be rewritten in many ways, but one popular preferred method is user stories.

In agile projects, the term "requirements" is hardly ever used, which explains why we have used the word "feature" up until now. Requirements suggests more traditional projects, where up-front definitions of terms frequently lead to the delivery of features at the expense of what the business really needs, which is value. They may also only be able to deliver a subset of features because they have sacrificed scope in order to turn the project in on time.

The radical difference in an agile project is that it determines features in terms of business value.

As already mentioned, this can be expressed through user stories that are not specific to any one agile method. In the process of contributing user stories to the product backlog, the Product Owner should always consider the business value that a feature adds.

Sample User Story
Title: Currency Converter

Description: "I would like to be able to easily convert from one national currency to another so that I can review web catalog prices posted in another currency denomination."

Acceptance Criteria:

1. US dollar conversion to Euros
2. Euro conversion to US dollars
3. US dollars conversion to Canadian dollars
4. Canadian dollar conversion to US dollars

$ ➜ €　　€ ➜ $

$ ➜ $ CAD　　$ CAD ➜ $

User stories should not be meticulously detailed written documents. It is crucial that you try to fit the description of a user story onto one 3" x 5" index card. If you go over, rip it up and start again. The purpose of user story is to prompt a deeper conversation between Product Owner and team, which are to occur daily during the sprint.

Even if software tools can afford more room than an index card, this helps build bad story writing practices. To establish good habits, we recommend first starting with note cards for user story management before moving to a tool, such as TestTrack.

ESTIMATING WITH STORY POINTS

If you choose to estimate using user stories with points, then the term used is story points.

As was illustrated earlier, estimations for user stories will occur iteratively throughout the project. In release planning, the Product Owner will discuss new stories that have been added to the product backlog in combination with the objective for the sprint or release. Should it look probable that any of these new stories will end up in the next few sprints, your team should stop in order to accurately estimate within the release planning time frame how many stories can be achieved.

3, 2, 5, 5, 8

EXAMPLE: **Our currency converter user story is projected on the team room screen, allowing everyone an opportunity to select their story point estimates using the Fibonacci sequence. Once everyone has selected an estimate, all of our story point estimates are revealed. We get the following numbers: 3, 2, 5, 5, and 8.**

The team and the Product Owner then discuss each new story until they have a closer appreciation of the user story in order to accurately estimate its size. The size estimate ought to be relative to other user stories in the deck.

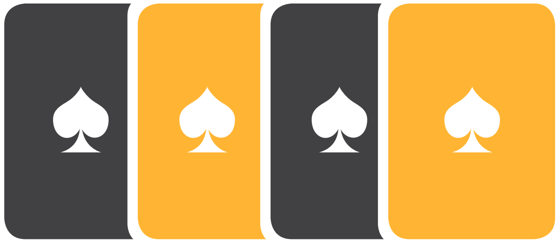
One reason why estimating in hours is not encouraged is because relative sizing actually tends to reduce the pressure of achieving precision. User stories estimated in hours, when it comes to exactness or predictability, might carry an aura of unfulfilled promises.

PLANNING POKER

A particularly useful means of building estimates is to play Planning Poker.

In Planning Poker, your team uses a deck of special cards with a number sequence (like the Fibonacci sequence), with each team member being dealt a hand. Cards are selected that team members feel most closely approximate the size estimate that they have chosen.

Once every team member is ready, they reveal their card to the other players, which means if there are outliers or a wide range of numbers, Planning Poker makes differing opinions more obvious and encourages a positive discussion that addresses such differences and imbalances.



If you change team members, this may change how stories are estimated and even affect the number of points your team can deliver in a sprint. Say two men, Steve and Ray, exchange teams and Ray is unfamiliar with his new group. It may take a couple of sprints before Ray can get in sync with this team's particular appreciation on sizing stories. Also, should Ray be unfamiliar with the code the team is working, the team should decide to take on fewer story points in the next few sprints until Ray gets up to speed. Relatively, both teams have the same workload.

HANDLING COMPLEX STORIES

Most plans are just inaccurate predictions.

~ Ben Bayol

User stories with unexpectedly numerous dependencies or many unknown variables should ideally be discussed during planning.

If certain dependencies are not well understood, or if there is a lack of specificity over what the story should accomplish, then estimates will most likely be longer to account for these extra complexities. Without sufficient information, a team will not be able to estimate accurately until the story is better explained. It is essential in this situation that the Product Owner retain sufficient stories in the product backlog for the sake of team discussion.

If a user story is too large to estimate or complete in one sprint, they are referred to as epics. Epics are placeholders for stories that shall eventually be spun out from the epic proper. Product Owners can use epics for planning purposes, but an epic will need to be broken into smaller stories before the team even attempts to make their estimation. Once the epic is broken down, it is typically discarded.

ARRIVING AT A CONSENSUS

After collating the team's story point estimates, it is necessary to build a consensus on what the ultimate story point should be for this particular user story.

This will allow you to bring up any story point estimates that seem excessively higher or lower than the average and consider the opinions of the team members who gave estimates outliers as to why they think this user story is any easier or harder than the other team members. They may have not fully understood something, or they may have noticed a problem that no one else has considered.

EXAMPLE: For our prior user story concerning the currency converter, the estimates outliers were 2 and 8. Some team members realize that there is more complexity when discussing the outliers than they thought previously, while some others realize there is less complexity. They all resubmit story point estimates. The results change to 3, 5, 3, 5, and 5. Therefore, the team decides on a final story point estimate of 5.

When the project commences, you should establish a means of determining how your team will proceed, even if there isn't a unified opinion. Project collaboration means functional teamwork, but it does not mean there has to be 100 percent agreement on every single decision. Before any situations arise, you should decide how to move forward as a group even when you can't achieve a complete consensus.

Fist of Five: Fist of Five is a quick technique for building consensus that allows team members to submit a vote between zero and five. Team members can then hold up the number of fingers on one hand, after conflicting story point estimates have been discussed, as an indication of how much they agree with the resulting decision. As a scale, a fist would indicate complete disagreement, and five fingers would show full support. Raising three fingers means that although there are reservations, the team member does not object to carrying out the decision.

ESTIMATING TASK HOURS

During the final stage of release planning, stories are selected by the development team members along with an agreement to deliver these from the product backlog in the current sprint. These stories will then become part of the sprint backlog for the current sprint.

Once the team identifies which stories will go into the sprint backlog, they can start sprint planning. This is where the development team will dismantle stories from the sprint backlog into separate tasks that constitute the story.

Here are some example tasks:

- build customer class
- migrate customer billing data
- add customer name field to database

Each task is then assigned an estimate in hours. As with estimating user stories, estimating task hours is handled by the development team.

During team planning, the Product Owner must be on hand. Once sprint planning starts, however, the team should have sufficient information to identify tasks to get started on. There is no need for the Product Owner to "handhold" while the team splits stories into tasks. This time can be used more productively elsewhere.

Ideal Days

Ideal hours are the units used for measuring task estimates. They are based on the idea of an ideal day, which simply implies a team estimate for the time needed to complete a task in perfect conditions, assuming no interruptions and a plentiful supply of energy and alertness.

REMEMBER

A task should hypothetically take no longer than 16 hours, or two working days, because smaller task estimates increase the overall understanding of the problem. From this, we can tell that a particular testing task, estimated at, say, 80 hours for a two-week sprint, would not have been broken down into the types of testing that needed to be done.

AVOID ANALYSIS PARALYSIS

If one is estimating either task hours or stories, it is vital that the whole team is aware that a law of diminishing returns exists regarding estimating time.

For user stories, if the team cannot choose or split the difference between a point size of five and eight, they should presume the worst and take the more conservative number, which is eight. In the matter of tasks, until they reach a natural stopping point, stories should be split into tasks and estimate hours. Tasks can be added or removed as the sprint progresses. Hours must be updated regularly, too, so as to afford the opportunity for retakes, redrafts, and rewrites.

Estimating in a Nutshell

Everyone has problems with estimation at first, but that is no reason to fear. Here is what we have learned:

- On agile projects, estimating takes place iteratively: for features at the start of each sprint and continually throughout the sprint for tasks.
- In the preliminary, high-level estimate, make certain you include everyone—in particular, the customer representative or Product Owner.
- To help in determining the difficulty level of each user story, assign story points.
- When estimating, make allowance for dependencies, and ensure that each user story is broken down into its component parts, or tasks.
- Stories break down into tasks, and the number of hours can be estimated for each task accordingly.
- Build group consensus on story point estimates so that work moves forward with a unified team.

PO	-	PRODUCT OWNER
SM	-	SCRUM MASTER
BA	-	BUSINESS ANALYST
DVP	-	SOFTWARE DEVELOPER
TST	-	TESTER

4. AGILE SUCCESS IS ABOUT BUILDING THE RIGHT THING: RELEASE & SPRINT PLANNING

“Strategy without tactics is the slowest route to victory. Tactics without strategy is the noise before defeat.” ~ Sun Tzu

RELEASE & SPRINT PLANNING

First ask yourself: What is the worst that can happen? Then prepare to accept it. Then proceed to improve on the worst.

~ Dale Carnegie

Let's review the progress of your agile project so far:

1. *Your product backlog is populated with features*
2. *High-level estimates have been made for a series of story points*

You are now ready to plan how you will deal with the project proper.

Release Planning vs. Sprint Planning

In the agile workplace, planning occurs on two distinct levels:

- Release planning focuses on the longer-term goals for the project as an overall strategy.
- Sprint planning deals with the specifics of each sprint. The intention of sprint planning is to generate a full sprint backlog.

Both of these planning phases typically take place on the first day of every sprint.

Once you have finalized your release and sprint planning, you will have produced a clear guide of your product's releases. We will now make an even more in-depth investigation of these two crucial agile activities.

Release Planning

Following the Product Owner's composition of the backlog of user stories, the whole team should meet for a release planning session. Over the course of this meeting—typically four hours—your team examines the project's strategic goals, commits to the goal for the current release, updates goals for future releases, and allows story point estimates for stories within the product backlog before creating a schedule of the sprints in the existing release.

Any release plan should begin with your prioritized and estimated product backlog, which should be a clearly identifiable synthesis of business value and delivery capability. The release plan sets down your release date and the number and length of sprints it will take to get there. Release planning, which should involve an estimation of new features, only ends with understanding precisely which features will enter the sprint backlog for the present sprint.

Release Planning Checklist:

- Product Owner informs the team as to any revisions to the release plan
- Current release/sprint goals are examined by the team
- Any new features in the product backlog are discussed and sized by the development team
- The development team selects features that they settle on to deliver in the current sprint

HELPING PLAN RELEASES BY USING THEMES

While it might seem that priority is the only thing that matters in release planning, other factors should also be considered. Common sense dictates that work on user stories with related functionality is comparatively faster than working on features that are littered at random throughout the overall application.

Quite simply, this kind of focus is more efficient because when you move from one functional area to another you must necessarily adjust your context. This switch then requires a period of adaptation for your team, which slows down productivity. By grouping work into functional areas, you reduce this context switching and speed up progress. If you also have multiple Scrum teams, then they can divide features by functional area building the same project.

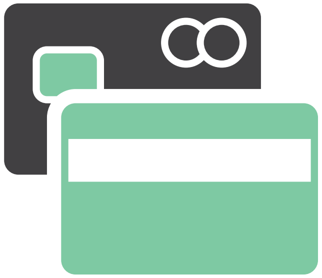
Functional area grouping is also much more efficient because themes help categorize the backlog, making items rapidly searchable due to their related functionality.

For this reason, grouping user stories by theme can increase efficiency.

EXAMPLE: **Say our customers find it difficult to find the data in the system that we are building for them, specifically that which is necessary to complete their weekly reports. A potential solution might involve user interface combined with changes to existing screens and a fresh business rule for automatically calculating status. If a report usability tag is created for items in the backlog pertaining to the fix, this will make it easier to construct a sprint to address the issue. The priority of the stories in the report usability theme into the product backlog is determined by the Product Owner, and the team specifies the stories they agree on to complete in the sprint.**

Themes and the Product Owner

Before the release planning meeting can even start, the Product Owner needs to establish themes. This will help the meeting zero in on the most significant issues concerning the value of the product to both the customer and the business. From the perspective of the Product Owner, themes should normally be attributed a business value—they seek to determine the benefit to the business or customer.



Consider the following three valid themes:

- Meet compliance
- Provide online payment options
- Improve administration

The Product Owner's view should be focused through the lens of these themes, but all the stories in a theme will not be equally important. Nor does the delivery of a theme's benefits require the completion of all its stories.

SET RELEASE GOALS

By commencement of the release planning meeting, the Product Owner should have concluded prior research so that all release goals can be brought to the table. These goals are not rigid, however, and will be modified should the team have certain insights that improve on any set of goals or if they lack the confidence they can deliver on the original goals.

The team must understand that value to both customer and business is the aim of any release goal, and that the Product Owner and management both fully understand the obstacles that must be overcome to meet these release goals. When the release planning meeting ends, all parties should have agreed on the goals for this release.

To ease release planning, the following tasks are needed. You will be required to:

- Select a desired sprint length. If your team has just begun working on the project, they will need to choose a sprint length. This can vary from one week to six weeks.

You must remember to remain consistent, though, since any natural operating rhythm that the team establishes for product releases will hold for the remainder of the project. Consistent sprint lengths also help stakeholders outside the team to determine the period of time involved in planning.

You should be mindful of the following factors when guiding the team to determine their initial sprint length:

1. Your team's experience and attitude toward agile projects and practices.

2. Your team's ability to deliver based on certain factors (such as their skills or relative maturity) as well as other organization factors, like management support and environment availability.

3. Any known dates that would affect the delivery of the work, such as release dates.

Always recall the Agile Manifesto Principle: "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." Your team should always seek new ways to deliver software faster but without compromising quality. Sprint lengths of five to six weeks could indicate room for improvement.

SET RELEASE GOALS

It is a mistake to look too far ahead. Only one link in the chain of destiny can be handled at a time.

~ Sir Winston Churchill

- Set target release dates. Once the sprint length has been calculated, the team can set target release dates. Release dates could be affected by external factors, such as trade shows or competitors, or indeed contractual needs; or internal factors that might include the commencement of a new fiscal year or reinforcing a reputation for consistent releases at regular intervals. For example, say your team specifies a four-week sprint schedule. This means that if the team decides to release every sprint, they have decided that they will be able to deliver code to production every four weeks. Alternatively, if the team chooses to release every two sprints, then they will deliver code to production every eight weeks. Your release plan needs to be built, in this manner, around a consistent schedule.

- Tag the release with a goal or goals. When the release target dates have been determined, the Product Owner can tag release goals to each release. Themes or epics could represent release goals, but these should not be interpreted as hard commitments by the team.

Even though a team might set releases at every two sprints, at the end of every single sprint, they will need to produce releasable code. At the end of the first sprint in a two-sprint release cycle, the team must have production-ready code.

Though Current sprints will have implementable features associated with them, release goals attached to future release dates should not. Each team will plan on a sprint-by-sprint basis, so it naturally follows that future sprint features can't be known until each sprint is planned.

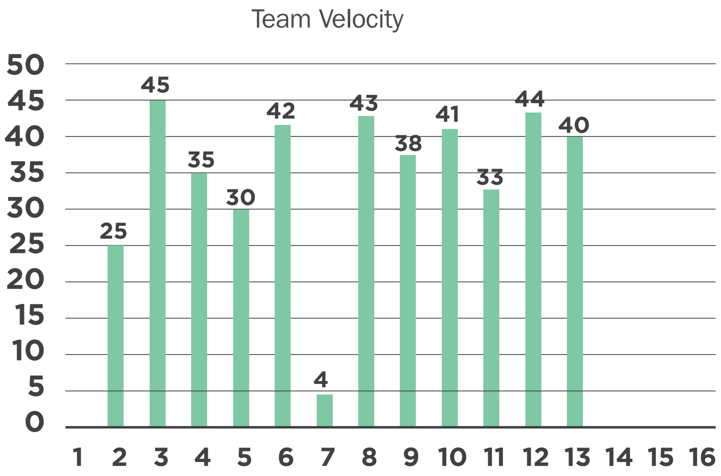
The benefit for agile teams is that a consistent release schedule sets a regulated rhythm for the project. This means that everyone involved with the project, both inside and outside the team, can predict new releases with a high degree of certainty.

CALCULATING VELOCITY

In the previous chapter, we covered estimating, story points, and we also mentioned velocity, which you will recall as the average number of features or story points that a team completes during a sprint. Velocity is very helpful, as it is useful for calculating the completion date of a project.

After velocity is determined, the team can accurately plan the number of sprints until project completion. Fortunately, with a firm grasp on velocity, this determination is not complex.

EXAMPLE: **If the team's known velocity is 20 points per sprint, it would take them about 12 sprints to complete a project with a total of 240 points in it. 240 (Total estimation points to build out project) / 20 (average points completed each sprint) = 12 Sprints**



Sample Velocity Chart

If the project is already under way, velocity can be established by examining past sprints. If the team lacks an established velocity, however, then a range is applicable: between a best- and worst-case scenario for project completion. This range is applied by having the whole team estimate the maximum and minimum number of points per sprint on features that have already been estimated.

EXAMPLE (continued): **The above team may instead have estimated that in a worst-case scenario 20 points per sprint would be their minimum velocity, and 40 points per sprint as their best case, or maximum velocity.**

The next page demonstrates two alternate ways of visualizing this sort of variability directly on the graphs that you are already using, which normally includes only the average:

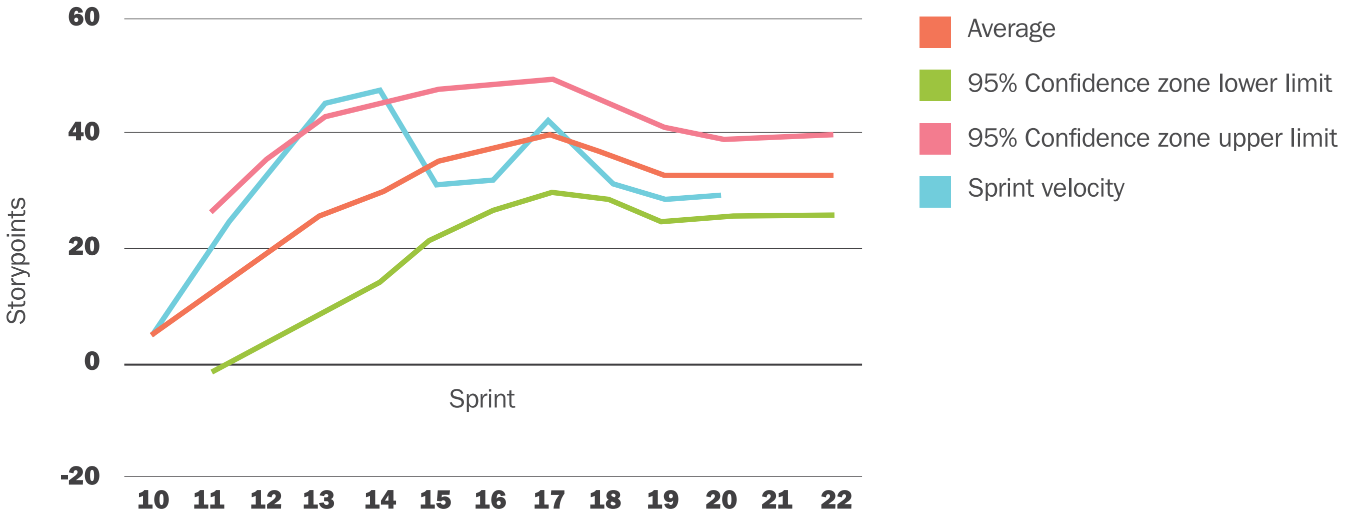
1. A confidence zone with an upper and lower limit graphed as part of a capacity chart

2. A pessimistic and optimistic forecast on the Burn Down Chart

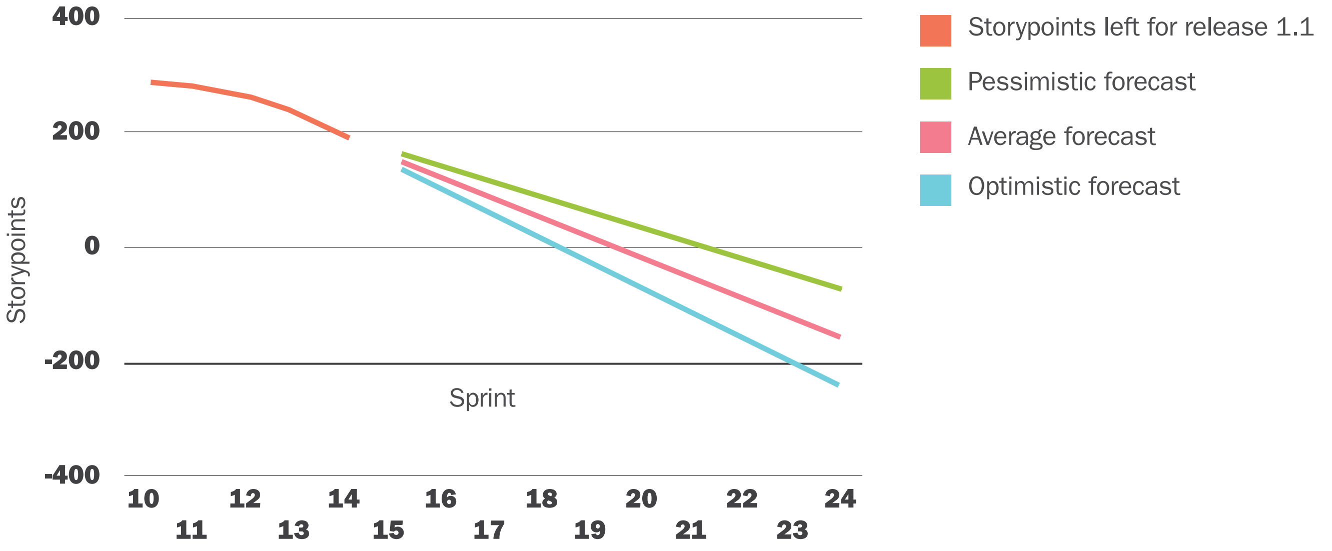
CHARTS PREDICTING BEST- AND WORST-CASE SCENARIOS

A trend is a trend is a trend. But the question is, will it bend? Will it alter its course through some unforeseen force and come to a premature end?

~ Alec Cairncross



Capacity Chart



Release 1.1 Forecast

SPRINTING TOWARD CHANGE

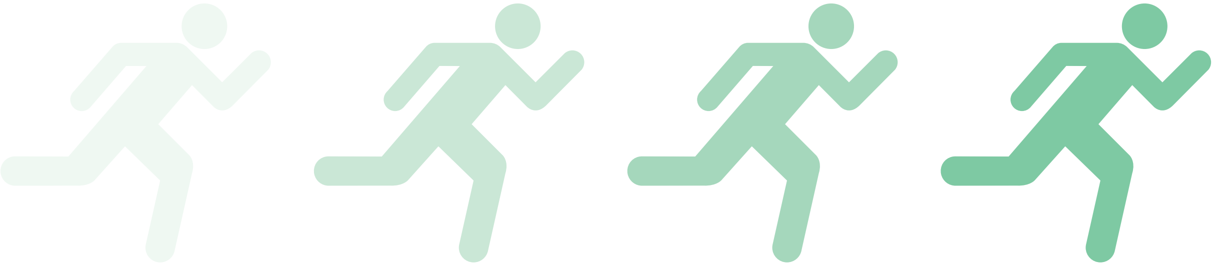
The initial release plan can be thought of as a rough draft. Even though you will need enough of a blueprint to get started, don't forget to continually revise and adjust your release plan on the go. This is integral to the agile process.

In other words, don't become upset if everything does not go exactly as planned, especially on the first couple of sprints. You will gradually develop a clearer picture of the team's actual velocity, which will then result in frequent revisions to the plan. Also, sprints should not be expected to deliver everything that you plan for them to deliver. Conversely, they might even deliver more. You will also need to account for staffing changes, customer changes, and other unforeseen problems that will affect your release plan.

The most significant reason, though, is that business and customer needs will inevitably change over time, often very quickly. It is vital to be flexible about future release goals to ensure you consistently deliver optimum value for the customer.

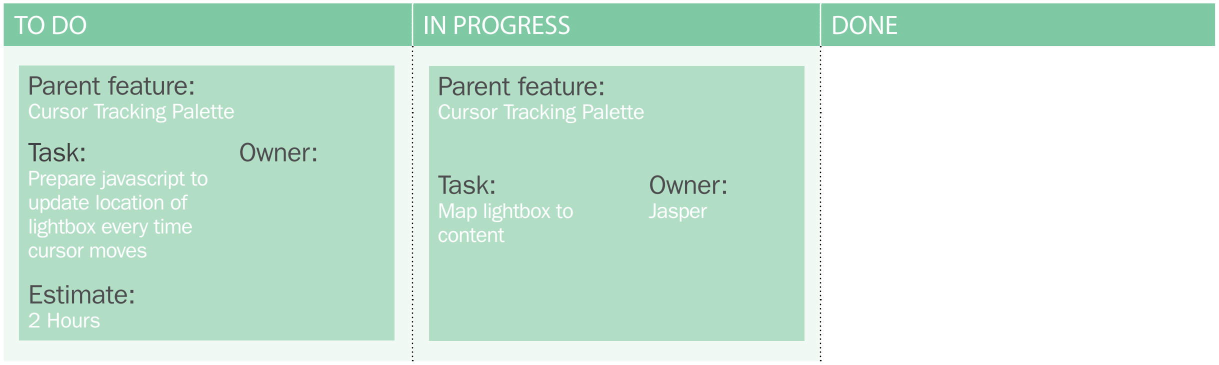
Zero In

If your team is new to agile, then planning for a Sprint Zero may be advisable. Sprint Zero is a sprint that takes place at the onset of the project, making it possible for a team to organize itself, set up a backlog, establish story points, set user stories priorities, and attend to technical and logistical issues. This Sprint Zero goal could even be to just build the backlog.



SPRINT PLANNING

In sprint planning, the development team breaks down the features that they have selected into tasks for the current sprint. The team will then seek to clarify and determine the number of hours needed to complete the task.



TO DO	IN PROGRESS	DONE
Parent feature: Cursor Tracking Palette Task: Prepare javascript to update location of lightbox every time cursor moves Owner: Estimate: 2 Hours	Parent feature: Cursor Tracking Palette Task: Map lightbox to content Owner: Jasper	

Observe that there is no name on the task card shown on the task board above. Until they move to "work in process," tasks are not assigned. Allocating tasks as they progress from "to do" to "work in progress" opens tasks to anyone and promotes team ownership for the work in the sprint.

Features should be broken down by the development team until they arrive at a natural point of conclusion. It would be pointless to attempt to identify every task since tasks will be added, removed, and updated continually as the sprint progresses. Project characteristics may affect the amount of time spent on sprint planning, but, generally a team should take four hours in sprint planning for a 30-day sprint.

The Product Owner need not stay for the sprint planning session. Since they have already discussed the sprint backlog features during release planning, the Product Owner typically does not need to stay with the development team as they deconstruct features into their respective technical tasks.

Once you have planned on both the release and the sprint levels, you will have a more complete release strategy.

HARDENING SPRINTS

So long as it is sensible, hardening sprints can be scheduled anytime in the release cycle. Hardening sprints should not address any new features, but allow for stabilization, bug fixes, and testing.

Here, a team can focus on chipping away at any technical debt that has accumulated over the life of the project. This results in a codebase that is easier to manage and maintain.

Hardening sprints is vital for large or complex projects. Technical debt will amass faster as the lines of code and number of components needing integration also increases. For larger projects, you will inevitably need to schedule time for hardening sprints.

Got a Question? Spike it!

Nobody is omniscient, your team included. By this, we mean there may be a backlog item that the team lacks information about and so cannot estimate in all honesty. If so, you might need to plan for a spike.

A spike is used to research a particularly problematic issue or technology to produce a precise solution for a business or IT question. Spikes can vary from a matter of hours to a matter of days, entirely dependent on the scale of the problem. Large spikes require a user story and can be estimated as if they were another story to be incorporated into a sprint.

Spikes cannot go on forever. You'll have to limit a spike's duration to hone the research effort and reduce the amount of time spent pursuing tangents.

A Quick Look at Release and Sprint Planning

Release and sprint planning are used to keep your project on track. Here's what we have learned:

- Release planning addresses your long-term, strategic business goals.
- Sprint planning deals with the particulars of each sprint. Sprint planning produces the full sprint backlog.
- Themes organize the product backlog so that you can easily find items with related functionality.
- Velocity keeps you informed as to the volume of work your team is able to complete per sprint and per release.
- For a new team to prepare for their first agile project, a Sprint Zero may be necessary.
- Hardening sprints allows teams to give attention to paying down the technical debt that has accrued during a project.
- Spikes are used to shore up any IT or business uncertainties through specific research.

5. AVOID WORK GETTING BOGGED DOWN: DAILY PLANNING

“The best preparation for good work tomorrow is to do good work today.” ~ Elbert Hubbard

HITTING THE TRAIL

Let’s once again recap the planning stages we’ve already covered:

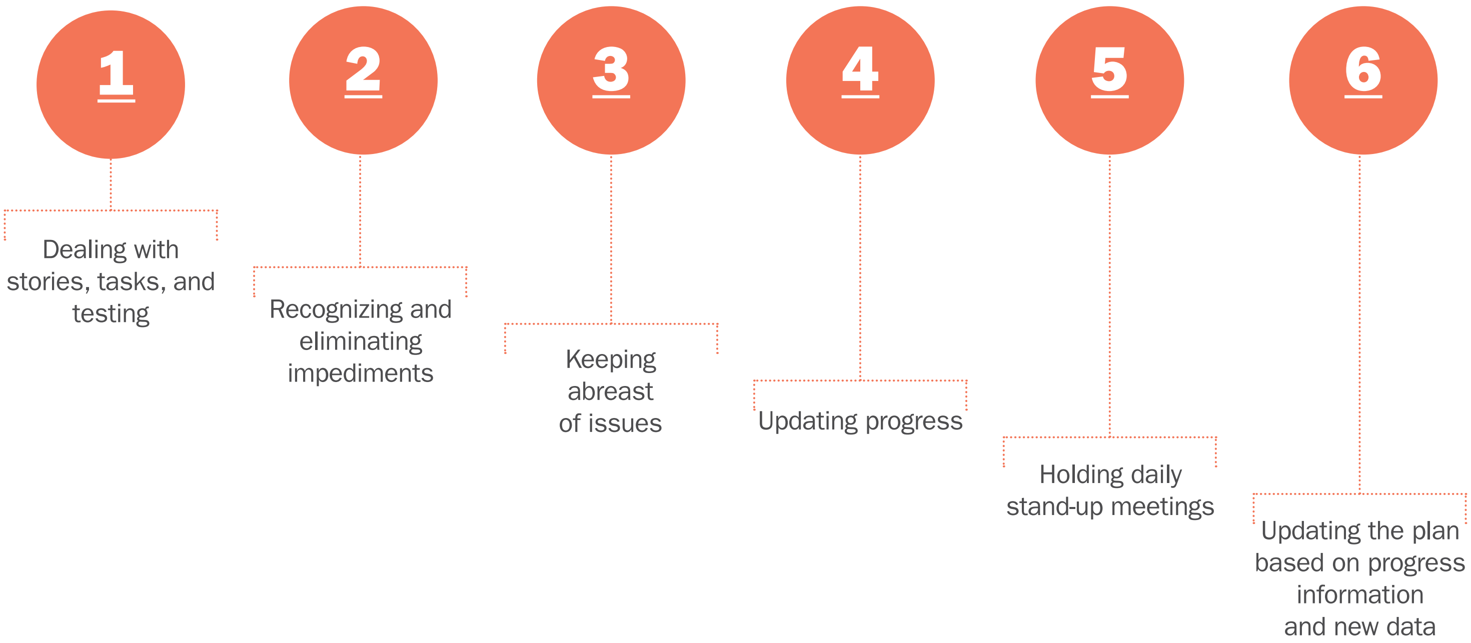
1. Populating the product backlog

2. Estimating efforts by points and hours

3. Planning the release and sprints

Now we’re ready to work through the rest of the sprint. Navigating these practicalities can seem perilous, but this chapter will provide guidance for your product build. Like every journey, it is simply a case of one step (or sprint!) after another.

From the shortest to the longest sprints, agile teams provide the benefit of an everyday schedule of improvement:



1 Dealing with stories, tasks, and testing

2 Recognizing and eliminating impediments

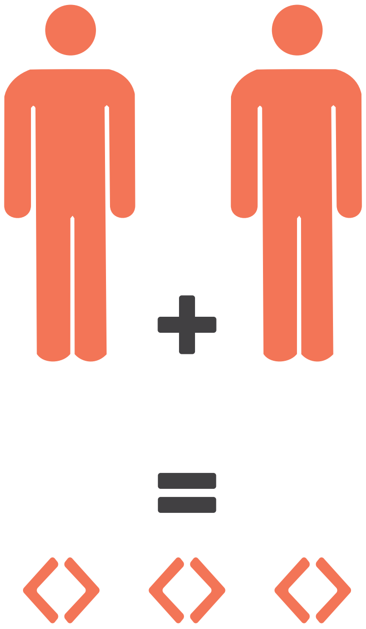
3 Keeping abreast of issues

4 Updating progress

5 Holding daily stand-up meetings

6 Updating the plan based on progress information and new data

WORKING THROUGH STORIES AND TASKS

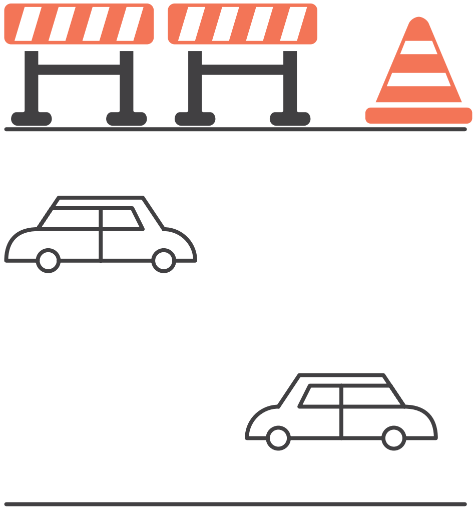


At the start of any new sprint, team members should prepare a list of which stories they have decided to complete in this sprint. At this point, a cohesive agile team should be more than capable of negotiating and conceding tasks to other team members.

It is absolutely imperative that team members be immediately vocal about any problems they encounter with a task rather than remaining silent until you start to run out of solutions. Another benefit is that if a team member completes a task earlier than expected or has certain skills and training that others do not, then that person can be depended upon to aid their colleagues, even for tasks they have not been allocated.

Pair programming is typically employed by many agile teams. Two developers get together to design and program as a duo. Advocates of pair programming have observed quality and productivity greater than the sum of any pair's individual contributions.

Another benefit of pair programming is code consistency, because two people have an observational advantage, each securing standards by acting as the other's safety net. You can learn more about pair programming in Laurie Williams and Robert Kessler's Pair Programming Illuminated.



Removing Roadblocks

It is inevitable that your team will run into roadblocks or impediments that will impede your project's progress.

An impediment is any issue that can reduce a team member's maximum efficiency and can be anything from the complex ("To test this report, I need data from another team") to a simple minor inconvenience ("I need a new mouse").

It is up to the Scrum Master to help eliminate team impediments. The Scrum Master is only as effective, though, as a team is forthcoming. To do their job properly, the team must communicate any difficulties as soon as they arise. The Scrum Master collects these problems at the daily stand-up meeting, the Scrum, and details new impediments and reports on solutions to previous impediments.

ENTER THE SCRUM MASTER:

WHO, WHAT, WHY?

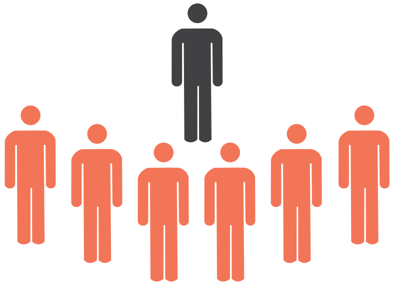
Although we have made frequent reference to the Scrum Master, this role has not yet been clearly defined. The Scrum Master's job is to make sure the team stays on the agile path. An expert in agile practices, they help ensure that team members adhere to the agile values at the center of the whole methodology.

When problems arise, the Scrum Master is there to keep the team productive, by maintaining focus on the agile method of dealing with those problems. The Scrum Master stays vigilant for external complications, while also monitoring the team to help facilitate communication and smooth interactions.

Open Up

Making progress obvious to everyone involved is at the heart of agile practice, from management and other stakeholders, and especially to the team itself. This transparency increases management's trust and confidence in a team, as well as enables their own capacity to self-organize and self-empower—twin aspects of agile that make for an unparalleled team.

Unfortunately, traditional development environments have the common potential to establish unproductive social habits, like the blame game and finger-pointing, among members of a team. These habits are counterproductive in any environment, but they will absolutely destroy the trust and openness needed in an agile project, making it so that people will not feel at ease to honestly discuss how their tasks have progressed.



The Scrum Master serves as a servant leader, vital in committing to a more open environment. The Scrum Master has to practice what he preaches and be a paragon of open behavior. For the Scrum Master, problem solving is an activity for the whole team, not a whip to castigate or penalize individual members.

HANDLING PRODUCTION ISSUES

The workday does not end until each member of the team updates their tasks with the number of work hours to completion. The number of task hours remaining in the sprint should be adjusted as tasks are removed or added, taking into account the hours for existing tasks as they advance.

EXAMPLE: Selecting the task of creating a new report, Barb estimated her time at two hours. Although she assumed that Joe had already created the tables for the report, she discovered later that he had not.
As a result, Barb must add "create tables" as a new task in the sprint backlog.

Update Your Progress

Under agile methods, code goes into production faster than it would in traditional development practices. Production code should be maintained even before the project has reached completion. How do you take care of production defects? The responsibility lies with whichever team is handling maintenance support and depends on any defect's degree of severity. Should a different team be in charge of maintenance support, then the project team is free to stick to project-related work.

If, however, your project team has to address maintenance support, you will have to factor extra time into your sprint planning to compensate for any emergency production issues. This potential grace period means that the team will have plenty of time to handle issues that require immediate assistance, as and when they occur. When a production issue is not an emergency, it is up to the Product Owner to establish the priority of every defect and write them up in the product backlog (just like another feature).

Setting up a triage process for production issues enables a team to stay on top of bugs without it interfering with their work schedule. With sufficient planning and prioritization by the Product Owner, new or existing issues can be assigned to future sprints, meaning that the team can stay focused on the features and tasks that are most valuable.

NEW

NEW FEATURE REQUESTS

Bugs are not the sole issue likely to occur during production. Customers will no doubt suggest new features or improvements for already existing features. Your team can actually treat these issues through the same framework that has been set up to address defects.

When the software does not work exactly as expected, users might even describe a feature request as a bug. It is essential that production issues are accessible to the Product Owner. An effective Product Owner always plans ahead to preserve a consistent flow of continuity from sprint to sprint. The Product Owner has to anticipate and plan the next sprint before the current sprint is over—planning new features, processing user stories, and composing backlog priorities.

Daily Stand-up Meeting

A routine of stand-up meetings, known as the daily Scrum, is a pivotal agile practice that assists communication and means that the team remains up-to-date on a sprint's progress.

These meetings should be scheduled daily for 15 minutes or less. Because the goal is for the team to update their status quickly, everyone remains standing during these meetings. (That is why it is called a "stand-up meeting.") Schedule it for a time when the following people will be able to attend regularly:

- Team Members
- Product Owner
- Scrum Master

In addition to these key people, the meeting is open to anyone else to come and listen in, but these visitors are not allowed to speak during the Scrum. It is certainly not a status meeting for management.

Every team member should be prepared to provide a ready answer to the following questions:

- What was your activity yesterday?
- What are your plans for today?
- Is anything obstructing you?

You may make the mistake of thinking the daily stand-up is for the Scrum Master. Rather, it is for the team, and they should talk to each other instead of reporting directly to the Scrum Master. The job of the Scrum Master in these daily stand-ups is primarily to listen, offer guidance, and find ways to smooth the team's progress.

KEEP IT SHORT AND CONCISE

Daily stand-up meetings are not designed or intended for detailed discussions surrounding scope, design, or similar project details. If deeper discussions on these themes occurs during a daily stand-up, they will be acknowledged and slotted for further discussion outside the daily stand-up meeting. This allows the daily stand-up meeting to stay focused on its primary goals:

- Assessing the team's progress from the day before
- Identifying any impediments that are preventing the team from making the forward progress necessary to successfully complete all the work originally planned for the sprint
- Confirming the work that will be completed that day

Your team's daily stand-up could indicate major problems if:

1. It lasts longer than 15 minutes
2. Participants sit down
3. Lengthy discussions ensue
4. It doesn't let you know if the sprint is on track

Making Progress Visible

When the team reports on progress, it should be easily understandable and readily available for other project stakeholders. Clarity is of the utmost importance to agile teams, and clear communication is the key. Two means of displaying progress are Burn Down Charts and Task Boards.

The Scrum Master needs to ensure that these key progress indicators are visible to keep everyone up to speed.

The sprint Burn Down Chart demonstrates to stakeholders the amount of work that remains in the sprint. By comparison, the task board is the best gauge of progress for teams rather than the Burn Down.

The Task Board shows which team member is tackling a task and the task's status: Not Started, In Progress, or Done.

These activities keep your sprints on track, so let's look at each in more detail.

SPRINT BURN DOWN CHART

A sprint burn down chart displays the work remaining in a sprint as graphical information. The amount of work to do in the sprint backlog is shown on the vertical axis, and each day of the iteration is represented on the horizontal axis.

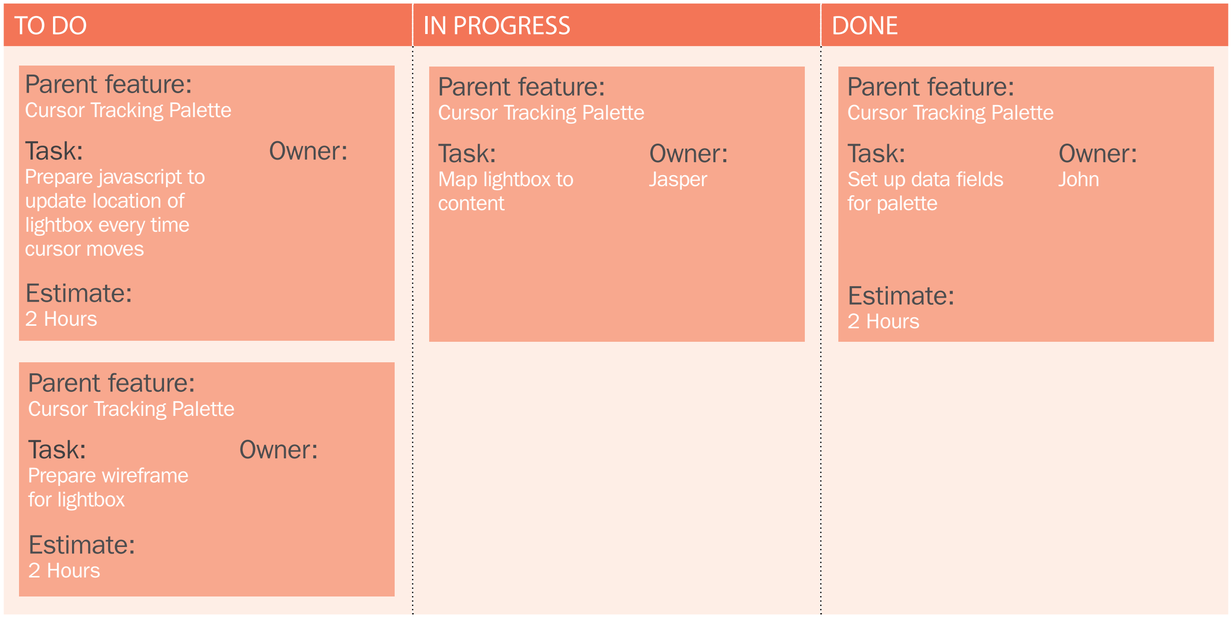
In a best-case scenario, an equal amount of time is burned down across the team each day, and at the end of the sprint the chart will have burned down to zero.

However, if your team members are honest in reporting their remaining task hours, then the line will usually fluctuate each day both above and below the ideal burn-down rate as it moves toward zero.

Burn down charts are covered in greater detail during discussions of metrics later in the Field Guide, along with visual examples.

Task Board

The task board displays the progress of stories by graphically showing the status of tasks needed to complete a story. These are displayed in columns across a board under headings such as not started, in progress, or completed. This gives stakeholders an idea of who is working on what and keeps them informed of the status of all items. For an agile team, it is an important (if not the most important) information source, or radiator. The task board depicts the progress an agile team is making in terms of making progress on the tasks associated with a story. It is the most instantaneous information radiator beyond the scrum, which makes it great for short-term information, though other radiators may provide more context.



TO DO	IN PROGRESS	DONE
Parent feature: Cursor Tracking Palette Task: Prepare javascript to update location of lightbox every time cursor moves Owner: Estimate: 2 Hours	Parent feature: Cursor Tracking Palette Task: Map lightbox to content Owner: Jasper	Parent feature: Cursor Tracking Palette Task: Set up data fields for palette Owner: John Estimate: 2 Hours
Parent feature: Cursor Tracking Palette Task: Prepare wireframe for lightbox Owner: Estimate: 2 Hours		

Most agile experts would recommend making a physical task board using sticky notes or cards taped or tacked on a wall and positioning it somewhere that is frequently visible to the whole team. Of course, the more modern solution would be to use a software tool that allows you to automatically generate and update your task board. You can even use a projector to display your digital task board on a workspace wall. This method has the added bonus of preserving the data for post-sprint analysis and reference.

WHEN THINGS ARE NOT GOING WELL

You may recall hearing Murphy's Law mentioned, which is a phenomenon applicable to your product as it is to any other endeavor. There are precautions, though, that allow you to limit the inevitable damage:

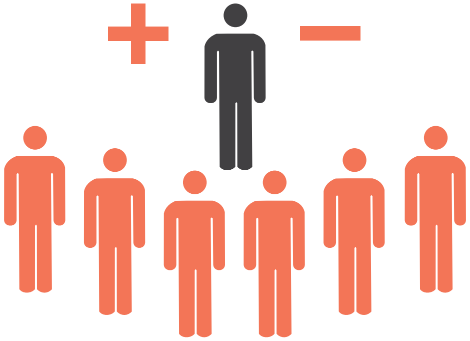
- Get the Product Owner involved ASAP. The sooner you raise the alarm, the more time you will have to solve the problem. In that way, there will be no nasty shocks.
- Involve the team. As with any development problem, the whole team needs to work together to find a solution. An individual problem affects everyone. So a developer might help with testing because it is no use to continue programming if his new code cannot be tested because bandwidth is not available.
- Communicate. Sprint status must be made available to everyone.
- Agree to drop an item. Your team needs to complete everything they agreed to deliver in the sprint, but if it becomes obvious that the sprint will not be completed on schedule, work with the Product Owner to drop from the sprint what is least valuable.
- Terminate the sprint. The sprint can be ended in a rare worst case by management or even the team. Decide to terminate the sprint only if the sprint goal becomes impossible or has changed so much that the sprint's yield will have little to no business value.

Checklist: Tasks by Role

We have covered many responsibilities in this chapter, so here is a checklist regarding tasks, divided by role.

Product Owner

- Participates in the daily stand-up meeting and sprint planning, review, and retrospective.
- Plans ahead to include features in the product backlog and organize and prioritize the backlog.
- Gives the team product development direction and is open for discussion and questions.
- Accepts or rejects features completed in the sprint.



Scrum Master

- Also involved in the daily stand-up meeting, sprint, review, and retrospective.
- Monitors and facilitates team progress.
- Acts as servant leader rather than a traditional manager.
- Seeks out impediments and neutralizes them.
- Aids in a team's understanding and adoption of agile practices.

Development Team Members

- Also involved in the daily stand-up meeting, sprint, review, and retrospective.
- Work on tasks broken down from features in sprint.
- Monitor issue queues.
- Deliver features committed to within the sprint to the best of their ability.

Daily Activities in a Nutshell

Basic agile activities occur daily. Here's what we have covered:

- It's essential to the project to communicate your status openly and honestly.
- Quality and productivity can be improved by pair programming.
- It is the Scrum Master's role to help remove impediments.
- Daily stand-up meetings foster colleague communication and help the team stay informed about sprint progress.
- The burn down chart and task board are vital progress indicators and are managed by the Scrum Master.
- To have time to resolve problems and impediments, talk about them as quickly as possible.

6. THE HEARTBEAT OF A MATURE AGILE TEAM: AUTOMATED TESTING

“Efficiency is doing better what is already being done.” ~ Peter Drucker

ITERATE WITH CONFIDENCE

Efficiency is doing better what is already being done.

~ Peter Drucker

Having worked through the previous chapter, you should now have the necessary agile skills to work on daily tasks, which means you can begin to outline your product's first iteration. You might even feel equipped for your next sprint. Before you start, though, you'll need to prepare for the possibility of breaking what you built in your last sprint.

Fortunately, agile takes this into account through automated testing, which lets you focus on how your new software features fit together piece by piece, making sure that each new sprint does not break features delivered during prior sprints.

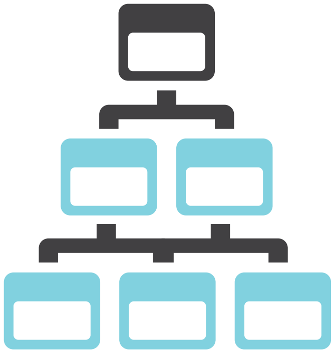
Test-Driven Development, or TDD, is a vital part of an agile user's toolkit. TDD is the practice of determining tests, even before you start to write code, and is a simple and practical means of improving software quality.

Automated GUI testing makes certain that an application operates correctly when accessed through the user interface. TDD and automated GUI testing complement each other perfectly, with TDD testing the code itself while automated GUI testing focuses on the user experience.

On top of this, automatic testing can also be applied to backend services, like databases, to guarantee that the parts of your application that are out of sight are functioning properly. When a user account is added on a new user screen, for instance, automated tests can be used to determine if the screen is functioning correctly and that the database tables were updated when the information was entered.

Customer acceptance testing is typically the primary concern of automated testing. Acceptance testing helps ensure that the whole product is operating with an efficiency equal to or even greater than each individual piece. Instead of the unit tests that are more often associated with the TDD model, deeply rooted in the actual code, automated tests are one step apart and thus one step ahead. This kind of service can be difficult to maintain, though, and can seem quite a trial to operate in the agile workplace.

In the transitional period, as your team adjusts to agile, you might think that the functionality to test seems to be in perpetual development, the formulation of test scripts focused with one eye backward. This can be avoided by making sure to create the right kind of automated tests and keeping their focus in the relevant areas. Automated tests should be a help, not a hindrance.



IS AUTOMATION WORTH THE EFFORT?

As we've just mentioned, automated testing can seem demanding for an agile workplace.

You might encounter such problems as:

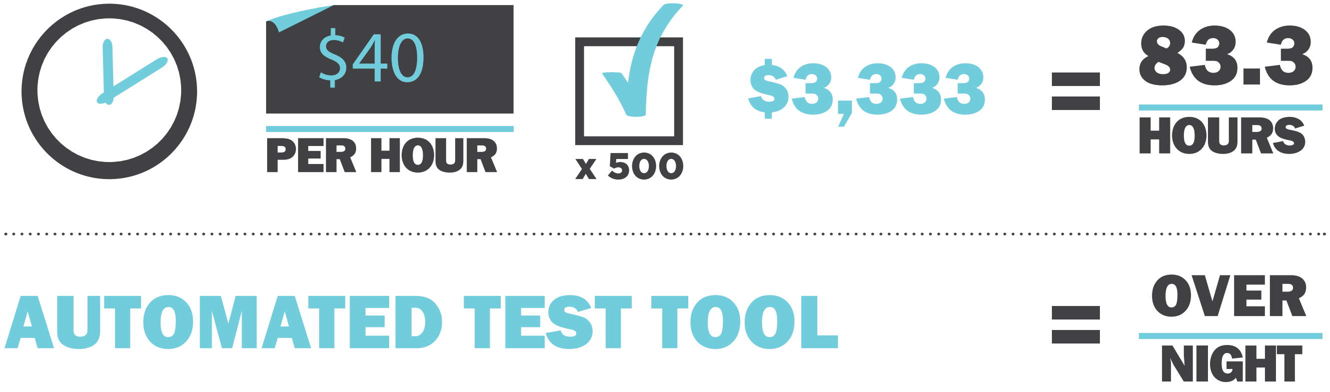
- Internal resistance from the company
- Some software development workplaces aren't prepared for automation
- Manual testers may not have the skills required to write scripts for test automation
- Certain financial costs are present related to training, coaching, tools, and hiring experienced staff

In the face of these problems, you might think that manually testing each sprint is easier, but this would be an error since you'd be missing out on the benefits automated testing provides for your long-term return on investment (ROI).

Automated scripts have a number of benefits over people:

- Consistently fast
- Tireless
- Never ignore test steps

The comparatively small investment in terms of test scripts and tools means that your testers can do a great job of testing novel or intricate parts of the application without having to worry about retesting the older ones. The ROI here can be expressed as "amount of time to run a test" multiplied by "the cost of a tester" multiplied by "the number of tests to run."



EXAMPLE: If it takes an average of 10 minutes to run a test and each tester must be paid $40 an hour, running 500 tests would cost $3,333 if each test cycle is performed manually. This would take 83.3 hours, or more than two man-weeks, for one tester to run! On the other hand, an automated test tool can do the same work overnight and will inevitably save you money and, more importantly, time. Additionally, you can test daily after every build rather than right at the end of the sprint.

A PRAGMATIC APPROACH TO TEST AUTOMATION

Efficiency is intelligent laziness.

~ David Dunham

If not guided by some pragmatic choices, test automation can take on a life of its own. Here is a list of recommended test automation practices to get you started:

- Test Clarity: All statements are simple enough to understand.
- Test Repeatability: Tests can be performed over and over without human interference.
- Test Efficiency: Tests take place over the course of a sensible amount of time.
- Tests are Self-checking: Tests deliver reports on their own results.
- Test Only What Is Necessary: Every part of each test contributes to the specification of desired behavior.
- Test Simplicity: Tests need to be as straightforward and uncomplicated as possible.
- Test Sufficiency: Tests address all the requirements of the software that is being tested.
- Test Reliability: Tests always produce the same result. Changes to their external environment do not affect these results.
- Test Traceability: Tests must be traceable both to and from the code they test and product requirements.
- Test Independence: Every test can be performed by itself or along with an arbitrary set of other tests and in any order.
- Test Failure Clarity: Every test failure specifies a particular piece of broken functionality.
- Test Maintainability: Tests need to be easy to understand, modify, and extend.

Many of these principles might seem tough to apply in a constantly shifting Agile environment. When choosing what to automate and what to apply each of these standards to, it might be a good idea to start with what you do not want to automate. For example, tests for exploring functionality and trying out edge cases are not exactly ideal for automation. If you keep your focus on core functionality, you will increase your automated testing ROI.

In choosing what to automate, you must look at these aspects of any development process:

Continuous integration involves bringing a team's code together as frequently as possible, once a day or more, to make certain the software keeps working in full even as changes are made. In its most basic form, with continuous integration, you can be positive that all your code still compiles and links. If combined with automatic testing, the value of continuous integration increases immensely.

Regression testing flags software errors by taking a modified program and retesting it partially, guaranteeing that errors are not established when fixing other problems. Unfortunately, this particular area of testing normally receives little attention.

Continuous Integration

Seasoned agile users perform continuous integration when making their team plan. TDD is your best option here, while your compiler and linker can catch basic “broken build” issues. Nonetheless, if you include a good subset of your automated tests, then you should be able to pay attention to issues inadvertently influenced by the current sprint’s updates.

In the process of choosing which automated tests to incorporate, you should include broad automated tests (like smoke tests) to make certain that the base functionality of your application is secure. You should use tests that address all the key application areas to make sure that testing performed in-sprint isn’t delayed.

EXAMPLE: If the application we are constructing needs users to log in before they can perform key actions, then we would need to include automated tests to make certain an administrative user can log in and view each main screen. We will not include security functionality until a later sprint, though, so there is no need to add detailed security tests to the current sprint.

Regression Testing

Regression testing involves more traditional automated testing, so the issue here is how to incorporate regression testing into the ever-changing agile environment.

Hardening sprints may be necessary, as the project dictates, to adjust for any gradual destabilization of your application. Over the course of each sprint, you may find yourself deviating bit by bit from your quality ideal as different parts of your system start to stray.

Through hardening sprints, you can refactor and integrate every part. By addressing errors as you encounter them—instead of paying back this technical debt right at the end of a release—you make your build more efficient and ensure better quality in the process.

You should make sure that mature or stable functionality remains at the heart of automated testing. You can use functional tests to exercise the most code and reduce the number of bugs that appear at later stages in the unchanged parts of your application.

REVIEW AT THE END OF EACH SPRINT

The best agile practitioners know that constant review and feedback are absolute necessities and apply this understanding to their automated testing. When each sprint ends, there will always be three key areas to review.

Initially, you must first consider what new functionality should be automated and added to the regression test or continuous integration suites. If you intend for new functionality to expand in the next few sprints, you will need to add a set of shallow tests, most likely to the continuous integration suite. After taking these preparations, it should be possible to turn those scripts into a deeper set that can then be used for regression testing.

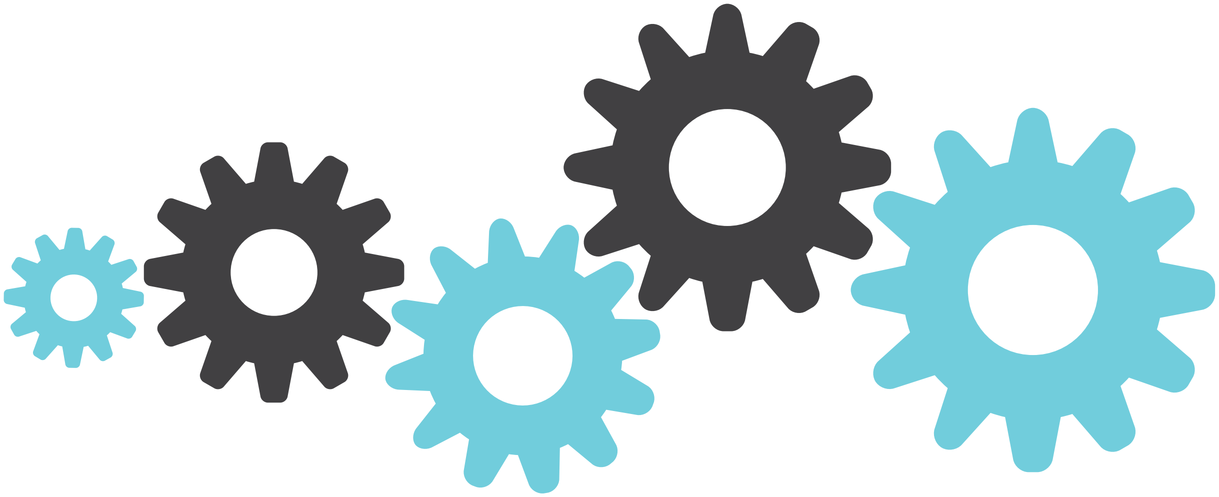
Later, you will have to contemplate the scripts in your automation suite that will need extra rewriting or even exclusion. It would make sense to rework or exclude scripts from test areas of functionality that are about to be changed extensively or ones that fail repeatedly due to changes from the current sprint. (You will want to be sure to reactivate those scripts when functionality is ready . . . a common source of failure!)

Finally, you should review areas that haven't yet been automated but that are related to considerable defects. Let's say you have had no regression or smoke tests concerning failed passwords, but an impromptu test has uncovered an aspect of the code that is sensitive to change. It would be sensible to add new automated tests to make sure that any future changes do not affect this particular sensitive area.

PLAN FOR AUTOMATED TESTING

One major benefit of an agile approach is the payoff that you get in terms of automation. When the whole team takes part in sprint planning, team members can focus on how and where to automate right from the planning phase.

By determining acceptance criteria, you also give yourself the option to consider what can and cannot be verified in automation tests. These acceptance criteria are consolidated into a solid acceptance plan. A weak plan composed of “screen should look clean and well-balanced” is hard to verify and worse to automate. Working together with the Product Owner, developers, and other stakeholders, the acceptance criteria and resulting acceptance plan can be crafted so that it can be effectively automated.

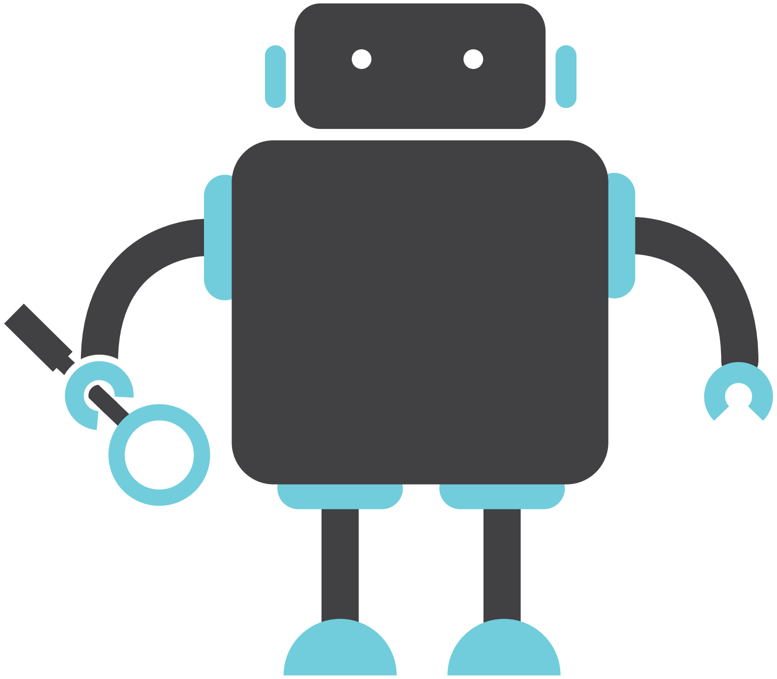


If every test requires you to rewrite after every sprint, you will definitely need to review your test automation process. Perhaps you are automating the wrong parts of the application, or maybe your scripts use “fragile” methods, like screen location, to find controls.

AUTOMATED TESTING IN A NUTSHELL

Automated testing means that agile developers can iterate self-reliantly. This is what we have covered:

- TDD is an ideal way to improve software quality.
- Automation contributes efficiency to your testing and offers a high long-term return on investment.
- Typically, Automated testing has addressed customer acceptance testing but can be used to verify data behind the scenes.
- Continuous integration makes sure that software keeps working even when alterations are made to the code.
- Regression testing assists in repairing issues as they arise.
- Automated testing likewise needs ongoing feedback and review.



7. DEFINING DONE: KEEPING OURSELVES HONEST

“Great is the art of beginning, but greater is the art of ending.” ~ Henry Wadsworth Longfellow

WHAT'S DONE IS DONE

Having just covered the importance of automated testing in the agile environment, we are led naturally to that developer's buzzword, the loaded phrase: "done."

It may not be immediately obvious why we should dedicate a chapter just to explaining the definition of done, because you may presume that the term is self-evident. "Done" is done, and that's that, right?

Wrong. Every member of your team will have their own subjective idea of what constitutes done. This is why a team must agree with every stakeholder on a precise goal line.

When typical waterfall projects finish, their predictive planning approach produces a customer sign-off phase that is almost unavoidably awkward. Consider a waterfall project that takes a year: done must be established a whole year ahead of completion. This issue is just one of the many predicaments inherent to predictive planning methods.

Unlike predictive planning, agile produces releasable features in a series of short sprints. This means that a team's idea of done is actually continually reexamined over the course of the "inspect and adapt" process. This reduces the possibility of discord or conflict when the sprint concludes because the whole team is forced to recall the meaning of done once every six weeks or less, in each and every sprint.

WHO SAYS WE'RE DONE?

It is the team itself that determines what done means, and they do so in release planning. This means that all those responsible for product delivery at the end of each sprint get their say.

If a team doesn't determine a precise condition for done, then differing subjective definitions will divide your team. This is to be expected: every team member will have their own point of view associated with the specifics of their job on the team.

While a Product Owner would set the acceptance criteria required for story completion, it is clear that technical team members are the ones with a real grasp on which technical aspects of the development process can be accomplished in the course of a sprint. External stakeholders will have an opinion on the sprint in the form of inputs and expect to get outputs in return, but these expectations should rarely determine what it means to be done.

"Done" Defined, Sort Of

Your team's definition of done should remain constant over the course of the project.

An organization new to agility may find that the precise notion of done changes over the course of several sprints, as your team gradually becomes more effective at the agile method of producing software and as the Scrum Master does their job of getting rid of obstructions.

Executable or production-ready code is the absolute minimum that an agile team should have produced at the end of every sprint.

No two agile practitioners will have the same description of what it means to be done. Every project is individual and idiosyncratic and when combined with a company's own particular culture, it is impractical to apply one form of done for every situation. Hence, it is vital that a team verifies their own conditions for done.

(?)

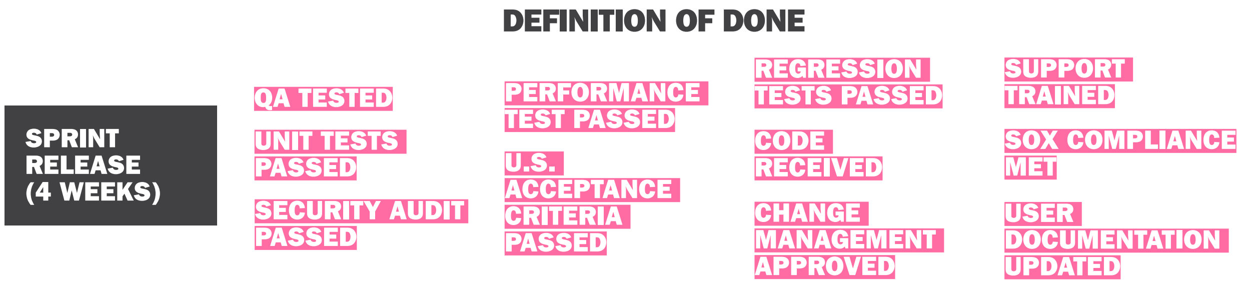
WHAT'S THE DEFINITION?

Now that you've explained why getting a group definition of done is important for a team, we must now examine what the actual definition should be. Determining done might take a while, especially if your team is unfamiliar with the Agile way of working with each other, as well as the product, as a part of an agile approach to your company's software development process. You should take a few hours to convene a meeting on the matter.

In the course of defining what done means for each sprint, your team should keep in mind:

1. The Product Owner's expectations.
2. What internal stakeholders need to get to done.
3. What external stakeholders need to get to done.

By having everyone in the meeting, it is important to make sure the proceedings are entertaining; perhaps book a conference room and use a whiteboard or sticky notes to produce a collaborative definition of done, as in the following example:



DEFINITION OF DONE

SPRINT RELEASE (4 WEEKS)

QA TESTED

UNIT TESTS PASSED

SECURITY AUDIT PASSED

PERFORMANCE TEST PASSED

U.S. ACCEPTANCE CRITERIA PASSED

REGRESSION TESTS PASSED

CODE RECEIVED

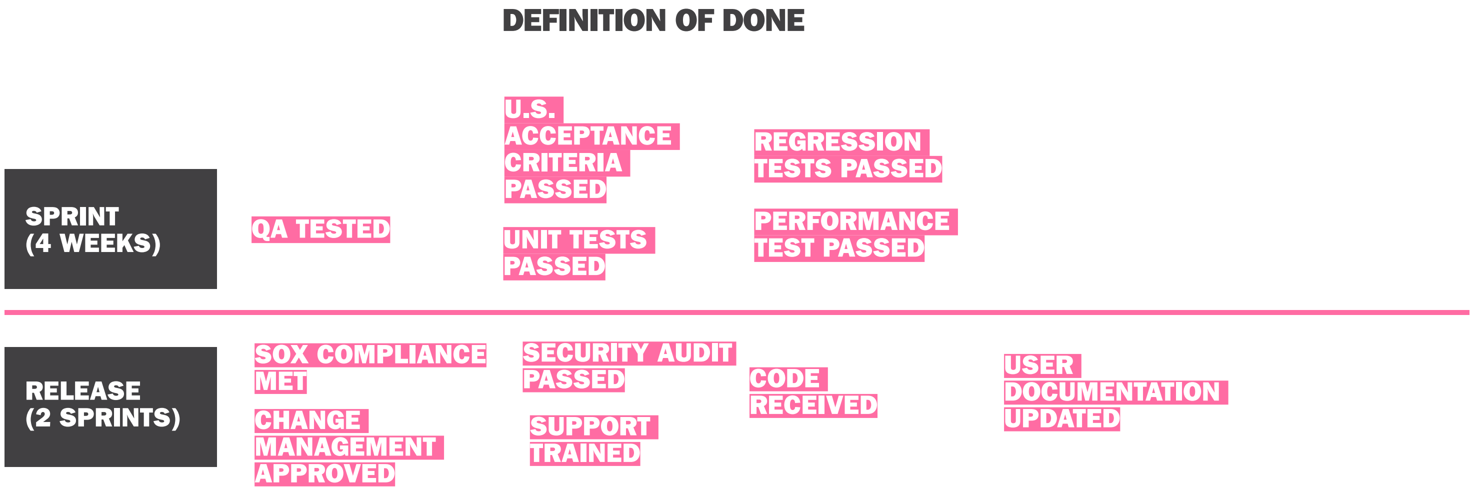
CHANGE MANAGEMENT APPROVED

SUPPORT TRAINED

SOX COMPLIANCE MET

USER DOCUMENTATION UPDATED

Release every sprint



DEFINITION OF DONE

SPRINT (4 WEEKS)

QA TESTED

U.S. ACCEPTANCE CRITERIA PASSED

UNIT TESTS PASSED

REGRESSION TESTS PASSED

PERFORMANCE TEST PASSED

RELEASE (2 SPRINTS)

SOX COMPLIANCE MET

CHANGE MANAGEMENT APPROVED

SECURITY AUDIT PASSED

SUPPORT TRAINED

CODE RECEIVED

USER DOCUMENTATION UPDATED

Sprint seperate from release

ADJUSTING SPRINT LENGTH

If you set your sprint length before classifying done, it will probably be necessary to alter the length of your sprint to get it in line with the team's designation of done. This may mean that a two-week sprint becomes a four-week one, but if your team genuinely cannot reach the agreed level of done in a sprint, you will need to isolate the obstacles that prevent this.

Communicating Done

Once a team specifies what done means to them, you will need to inform all stakeholders. Get your definition on a visible information radiator, such as a Scrum board or task board, and put it in a team room, the hall, anywhere that is frequently reviewed by stakeholders.

If your information radiator is disseminated through electronic means, be sure all stakeholders have access and are aware of how to see the latest reports on the project's status.

If something goes wrong, the visibility of the information radiator will make it so that people are aware of this. A natural human instinct is to want to insulate yourself (and your team) from spreading this information. It takes considerable trust to put all this information out there in a candid way, but it is crucial to do so if your organization is going to adopt an agile way of working. Never forget that a lack of trust is simply not the agile way:

By being open and clear about the project, you will foster trust within your team, between it and the whole organization, and even with external stakeholders.

Agile software development has the added bonus of highlighting a company's waste. As we have already discussed, the average sprint is four weeks, plus or minus two weeks. Sprints cannot be any longer than six weeks and have to produce releasable code by this deadline. If it takes longer to get to done, this is a strong indication that your development process is wasteful.

ELIMINATING WASTE

All activities that do not contribute value to your final product can be defined as waste and will need to be identified and removed. One of the most effective and proven means of identifying waste in any process, including software development, is using a technique called Value Stream Mapping (VSM).

Value Stream Mapping is a lean analysis method used to examine the flow of work through a particular organization. The objective is to reduce wasted effort by making the company “lean.” For a basic overview of the Value Stream Mapping technique, visit:
http://en.wikipedia.org/wiki/Value_stream_mapping.

Once a potential waste item is identified, you should adopt a means of finding the root of the problem, like the Five Whys. If you have ever used Lean Manufacturing, Six Sigma, or Kaizen practices, you may already be familiar with this term. In this particular method, you repeat “Why?” five times to illuminate the source of the problem.

DEFECT	REASON
Why 1: Why did the defect occur? ←	
Why 2: Why did THAT occur? ←	
Why 3: Why did THAT occur? ←	
Why 4: Why did THAT occur? ←	
Why 5: Why did THAT occur? ←	

ELIMINATING WASTE

EXAMPLE: "QA Tested" has been obstructed, and the team cannot get to done in the course of their four-week sprint. If you recall, a team needs to have production-ready code at the end of every sprint. Even though this team could up their sprints to five or six weeks, this still would not address the fundamental issue.

With the Five Whys, a Scrum Master asks "Why?" up to five times, which gives them some idea of what is obstructing QA:

1. Why can QA not test within one sprint? It is necessary to ask for the QA environment, which necessitates a three-week lead time.
2. Why is it necessary to ask for the environment? This is because all applicants share the QA environment.
3. Why is it necessary for there to be a shared QA environment? There are not enough QA team members to arrange and configure the environment.
4. Why can the dev team not plan the environment? Policy makes clear that code can only be deployed to the QA environment by team members.

In this case, after just four "Why?" questions, you can tell there is an organizational obstruction to QA testing this sprint's code. A Scrum Master, in charge of addressing these obstructions, will need to meet with the suitable managers to handle this policy and, eventually, have it adjusted or even halted.

WHY?

WHY?

WHY?

WHY?

WHY?

DONENESS IN A NUTSHELL

The way to get things done is not to mind who gets the credit for doing them.

~ Benjamin Jowett

Determining done will make sure that the members of an agile team, as well as other stakeholders, will know the stopping point for sprints and releases. Let's recap:

- A team continually reassesses what they mean by done over the course of an agile project.
- Due to done being reconsidered every six weeks at the latest, there is less possibility of discord when the sprint finishes.
- A team precisely defines done in release planning.
- If sprint length has been set before done has been determined, then the team will probably need to alter it.
- After done has been outlined, this description should be disseminated among all stakeholders.
- Eliminating waste will aid in getting a better definition of done.
- The "Five Whys" is a useful means of getting to the source of obstructions.

8. SHOWCASING FOR FEEDBACK IS CRITICAL: THE SPRINT REVIEW

"Truth is confirmed by inspection and delay; falsehood by haste and uncertainty". ~ Tacitus

AFTER DONE: WHAT COMES NEXT?

Perhaps you might think we are finished with Doneness Criteria, but your agile experience is far from over. You still have the sprint review to consider.

At the sprint review, you get yet another chance to check in with your team and assess the product that they are working toward. More than this, it gives stakeholders an insight, with "stakeholder" here referring to any person or organization affected positively or negatively by the product's outcome (according to the classification from the Project Management Institute (PMI)).

Given that criterion, the sprint review will have a wide range of participants . . . some of whom are familiar from earlier in the process and some who haven't had much direct involvement in the process. A list of possible stakeholders includes:

- The Scrum Master
- The Product Owner
- Development team
- Technical teams that collaborated
- The project sponsor
- Senior managers
- Executives
- External organizations

Don't leave attendance invitations to the very day of the sprint review. The Product Owner and Scrum Master should start to make a list of whom to invite to a sprint review before the project even starts. This list will then be adjusted as the team iterates. Proper visibility of the project depends on which stakeholders will attend the sprint review. Though it may turn out that it is only your team attending, always be vocal and offer the option to attend every review to all appropriate stakeholders.

FIND YOUR MARCHING CADENCE

If you have kept a consistent sprint length, you will have noticed that your team will start to establish a natural rhythm or cadence. If sprints are set for every four weeks, stakeholders can presume that the next sprint review will take place exactly four weeks after the one prior to it.

Just like the stand-up or daily Scrum, you will need to set a specific time and place for your sprint review. This reliability allows stakeholders to keep up: senior managers and executives will spend their days in and out of various meetings, so they are more likely to attend if the event is confirmed.

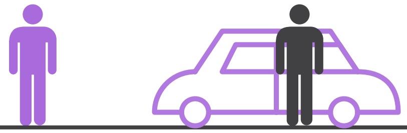
Who Wants to Drive?

No one person is in charge of "driving" every sprint review. The project would not exist without a dedicated team, so part of the agile process involves spreading the presentation work around. This helps promote the team's mutual effort on the project. Everyone has the opportunity to show off, rather than just leaving every demonstration to the Scrum Master or Product Owner.

In other words, every sprint review can be facilitated (or driven) by a different team member. This can be hard for those transferring from traditional hierarchies to the agile system, but it is important. Here are just a few of the benefits of this approach:

- Team members feel a sense of ownership
- The Scrum Master position as a "servant leader" is reinforced
- The team members become accustomed to leadership that isn't always top-down
- Management (and other stakeholders) gain better understanding of how many people are involved in the project

As already mention, the Scrum Master is a "servant leader," so it is not their duty to specifically apportion sprint review responsibility. It is up to the team to choose who is going to drive. The decision is more effective when arbitrary, though, so encourage them to draw straws or names: anything that seems fun.

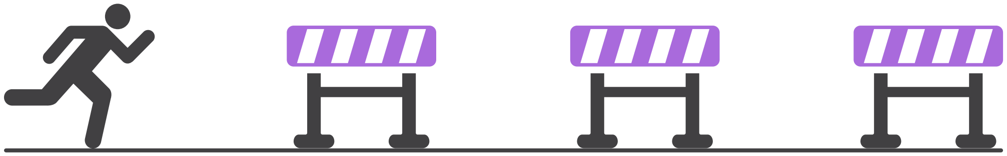


GET ON THE SAME PAGE

When your team accepts a user story into a sprint, they are obligated to the Product Owner to complete these user stories over the course of the sprint.

There may be a number of obstacles to completing user stories on time, such as the Product Owner not giving sufficient information when needed, developers allocating too much work, or other issues that are internal to the organization. It is crucial for the team to devise clear messages concerning unfinished stories to present to stakeholders at the sprint review.

Both the Product Owner and the team should be reading from the same page regarding the sprint review. The team should not face any surprises, so the Product Owner needs to admit or reject which user stories are to be discussed according to their acceptance criteria prior to the sprint review. The reverse is also true: the Product Owner will also need to know about any unfinished stories before the sprint review starts.



Should any user stories end up incomplete after the sprint, the team will need to disclose what they know to have gone wrong with total honesty. Agile is dependent on trust, so the team needs to be completely sincere to earn it. Should your team not know what went wrong, you must tell those attending the sprint review that the sprint will be examined in the retrospective. You must be certain to make sure that the retrospective results, as well as any associated actions or improvements, are made visible to all stakeholders.

THE SPRINT REVIEW STEPS

Once you have determined who is going to drive the meeting, gathered all the stakeholders, and everyone in accord the sprint review can start. Three things take place in a sprint review:

1. A concise summary by the Product Owner of sprint or release objectives.
2. A concise summary of a specific user story by a team member who will then demo it in the software.
3. The team fields questions about implementing the user story and takes on board anything that could be valuable to the retrospective.

The second and third steps should be repeated until every story in the sprint has been covered.

You will need to allow sufficient time to perform a sprint review. Typically, you would plot out four hours of meeting for every thirty days of sprint. This will need to be altered according to a project's needs, but it is a good idea to try and keep meeting lengths as consistent as possible for every sprint.

Completed Stories

With a physical task board, teams would normally tear up and discard the cards for finished stories after every sprint. This isn't very productive, though, since it prevents archiving. Your organization might undergo an IT audit or you might want to re-examine how you went about addressing a similar story in a prior project.

Physical means are available for preserving this data (such as filing your cards or taking a photo of your task board when the sprint ends), but a better, more efficient solution would be to use a software tool that can keep track of user stories and produce your task board and Burn Down Charts, as well as similar reports at the push of a button. By using a software tool, you secure the history of your stories, creating an easily accessible database to refer to when needed.

Electronic recordkeeping is really useful for teams that have to contend with industries that are heavily regulated. Software tools offer a flexible and pragmatic solution by collecting the kinds of data needed to pass audits and demonstrate accountability, all without interfering with the practice of agile.

These are called Application Lifecycle Management (ALM) tools. Some of the leading agile tools in this category are Rally, Version One, Pivotal Tracker, and Mingle. See the Resources Appendix for more information.

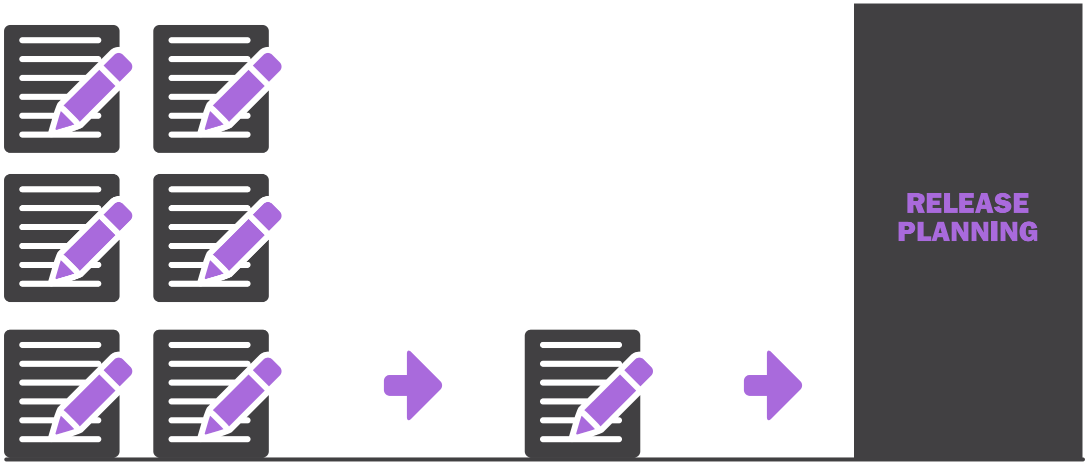
WHAT HAPPENS WITH INCOMPLETE STORIES?

When your team accepts a user story into a sprint, they are obligated to the Product Owner to complete these user stories over the course of the sprint.

Any story that remains in the sprint backlog once the sprint is over is transferred to release planning for the next sprint. The Product Owner will discuss various stories with their team, and they will decide as a group whether or not to include a story in the next sprint.

Your team will need to be aware of three things in their decision to keep developing any unfinished stories:

1. What business value does the story add?
2. What are the technical consequences of stopping work on it?
3. What extra support might be required to keep working on it?

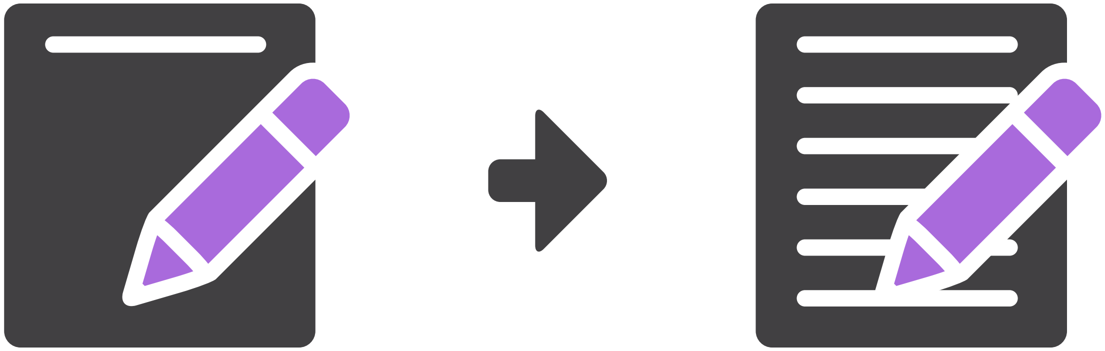


If your team is using story points to estimate user stories, incomplete stories do not count. No matter the issue, a team only gets points for the delivery of complete stories. Agile is unlike more traditional methods, where value is determined by the work allotted to a team. Instead, the team has the capacity to pick what work they will take on in each sprint. Through these means, they can develop trust by delivering the stories they have committed to.

SHOULD YOU REESTIMATE?

When planning for your next sprint, you may consider reestimating any stories that remain unfinished. There are many opinions on the matter, but reestimating unfinished stories from a prior sprint could be a waste of a team’s time and cause stakeholder confusion. Alternately, one benefit to reestimating the amount of effort left over at the beginning of a new sprint on an unfinished story is that it will discourage teams from starting stories that they can’t finish, because they won’t get credit for the unfinished effort.

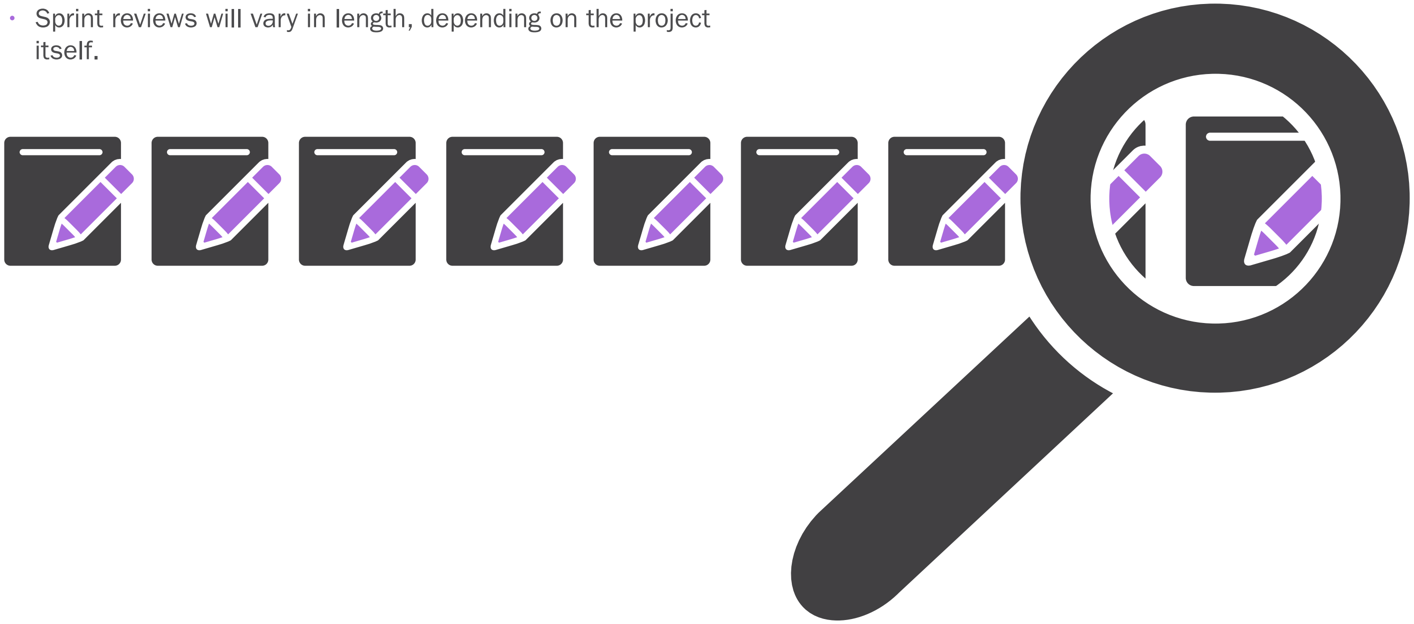
If the team did not get started on a story in the prior sprint, then there is no problem with incorporating it back into the product backlog and comparing and resizing against the other stories already in there.



SPRINT REVIEW IN A NUTSHELL

The sprint review offers yet another chance to inspect a team's progress and check on the product they're creating. We have learned that:

- "Stakeholder" is a term that covers any organization or person that may be positively or negatively affected by a project's outcome.
- If your team is to benefit from a cadence, your sprint length should stay consistent.
- Your sprint reviews will need to happen at the same place and time to avoid misunderstandings and to encourage consistent attendance by the less-engaged stakeholders.
- The person "driving" can change as frequently as every sprint review.
- Before the sprint review, your team should already be aware of the topics that will comprise this meeting. There should be no surprises.
- As a group, the team needs to decide on how (and if) they are going to present unfinished stories to stakeholders.
- Sprint reviews will vary in length, depending on the project itself.
- For future reference, use software tools to generate and archive your finished stories, as well as any other relevant data about your sprint.
- It is possible to take unfinished stories across to release planning for the following sprint.



9. TAKING TIME TO LEARN: THE SPRINT RETROSPECTIVE

"The creative process involves getting input, making a recommendation, getting critical review, getting more input, improving the recommendation, getting more critical review… again and again and again."

THE END OF THE ROAD

Once again, you might think you've reached the end of the line, pulling into your final destination with a completed sprint. After demonstrating your product at the sprint review, you may believe that the job is done, so you'll want to kick back and put your feet up.

But not just yet—there is still work to be done! Your sprint review might come right at the end of sprint, but a sprint is not over until the sprint retrospective (sometimes also called the "postmortem" or "lessons learned" in non-Scrum methodologies).

As the name implies, sprint retrospectives are normally held after sprint reviews and give the team time to pause and consider the sprint as a whole (which includes the review as well). This is not always the case, though, as conflicts of scheduling can occasionally mean that the sprint retrospective may precede the sprint review.

No matter which way your team decides to hold their sprint retrospective and review, they need to happen on the same day so that the team does not lose their cadence. Getting them both out of the way means that your team can get closure on the whole sprint and start looking ahead to the next one.

?

WHY A SPRINT RETROSPECTIVE?

Conventionally, retrospectives are held when the whole project concludes. While this may seem logical, taking a retrospective right at the close of the entire project produces a few drawbacks:

1. The time to fix or change anything is long past. Since the project is over, the only thing you can try and do is learn from any mistakes you may well have made and produce a wrap-up report to file. It's possible this might help out future projects, but it does nothing to help out the project just completed.

2. You have to contend with memory gaps too. Team members coming out from under the pressure of a job will probably have problems remembering what they did last week. Asking them to think back a year or more is unthinkable. If you wait until the end of your project to do so, there will no doubt be substantial differences between what the team recalls and actual events.

3. Energy and morale will probably be depleted. All are aware of the problems just listed, which can be dispiriting for a team if the retrospective takes place when the project ends.

Retrospectives at the end of every single sprint give your team a chance to touch base and work on the way they deliver a project. This allows the next set of work to be continually informed by the problems encountered in the most recent work.

The Last Inspection

In agile terms, a sprint retrospective constitutes the final stop in a whole series of inspections throughout the sprint proper. However, this time around the retrospective inspection offers an ideal occasion for a team to find ways to evolve and progress their use of agile methods. Compare these three benefits with the issues we have just listed:

1. You have space to make changes. Anything you encounter in a sprint retrospective can help out the following sprints and advances development for the product, the customer, and your project team itself.

2. Your memories will not be clouded by time. Your team only has to recall a matter of weeks rather than months or even years. As a result, any differences between recall and reality will be minimal.

3. Morale and energy will both be peaking. Your team knows that anything raised in the retrospective will improve the next sprint. Rather than being an exercise in futility, your retrospective uses your time to make a change for the better.

WHO'S INVOLVED?

At the very least, a retrospective will need to be attended by the core scrum team of the Product Owner, the Scrum Master, and development team, as well as any support teams (those who aided in completing sprint objectives but are not allocated to the project as a whole) You can include other optional participants, though, in the form of other stakeholders (i.e., managers) who may have something useful to contribute.

Teams and organizations new to agile might only have a retrospective with their delivery team. Just like a review, however, project stakeholders need to be extended an invitation, since the information that a sprint produces will be their concern as well. They will want reports, metrics, and the like. Should there be an issue, for example, with how a Burn Down Chart has been structured, then stakeholders will no doubt want to discuss this in the retrospective in order to make changes to the structure of the chart in question.

How Long Should the Retrospective Last?

As with the review, the length of a retrospective for a thirty-day sprint will take about four hours. Any retrospective will need to be adjusted for the specific requirements of a project, as well as the team's own capabilities. For any team unfamiliar with agile, retrospectives will probably take longer, but this should eventually diminish as they come to understand the process.

On the other hand, as the advantages of a retrospective become more obvious, the length might actually increase as your team gets more involved.

If more people are involved in a sprint retrospective, it will be necessary to take more time for your team to gather and assimilate all the data that the meeting produces. There may be issues with time constraints, where a retrospective becomes stagnant and does not offer any actionable results of value. You cannot have your team see a retrospective as unnecessary or pointless, so if time becomes a problem, you might want to split into groups to tackle each issue and then reunite and compare notes.

FACILITATING THE RETROSPECTIVE

While your team is unfamiliar with agile methods, the Scrum Master should lead a retrospective since he/she will no doubt have a greater understanding. Once your team comes to grips with agile sprint retrospectives, other members can be offered the option to lead. This promotes to your team the idea of buying in and ownership, which will lead to more efficient and productive retrospectives.

Whoever facilitates the retrospective, they want to be sure they begin by making sure everyone is on the same page about what happened during the spring. Next, the retrospective should collect data from the participants and use that data to drive insights into what should be done next. The decisions that come out of this meeting should be absolutely clear to everyone by the time the retrospective closes.

A retrospective needs to be both enlightening and, to some extent, entertaining and need not take an eternity to organize. Keeping candy on the table can keep up blood sugar levels. In addition to a facilitator's capacity to conduct an engaging dialogue, markers, a whiteboard, and some sticky notes are another simple means through which you can make the whole process tolerable and maybe even fun!

What worked well	What didn't work well	What still puzzles me
	Scope changed on batch archive without re-estimate	Estimate still not right
Structure	Not doing TOL before starting	
Spike before development worked well		
Pairing rotations - much better		

Though the example on the previous page uses sticky notes, a whiteboard could just as easily have been substituted in order to follow:

1. What has the sprint done well? What worked? Any interactions, processes, and events that were useful to your team and that they would like to see more of should be listed.
2. What hasn't really worked in this sprint? Things such as delays, obstacles, and broken processes should be listed so that your team can either work on or discard them.
3. Which actions will advance the process? Write a list of the actions that team volunteers, such as the Product Owner or Scrum Master, have agreed on to finalize and complete in following sprints. Rarely, some actions can be picked up by the whole team.

It is not simply a case of recording things that worked and things that did not. Anything that can be improved needs to be recorded too. This way, you make certain that these improvements are incorporated into your next sprints and releases. Agile does not just mean that you inspect—it also allows you to adapt.

Carry It Forward

Your team will determine actions and improvements that need to be applied to following sprints according to the information gathered in the retrospective. It is a good idea to start retrospectives with a recap of the actions taken from previous ones. By being reminded of the decisions (and their consequences) from the past, the team will be primed to move forward with this retrospective.

You should not be bothered if your actions from a prior retrospective do not produce useful results. They are somewhat experimental—figuring out how to improve an already existing process. Just like any experiment, you have to be prepared to learn from failure. The team might also recognize as an item any activity that was performed well in one sprint but should also point out if it has fared worse in a later sprint. This is fine, too, but if you see too much transition from "good" to "poor" on an item, you will need to investigate to find the root cause.

Should an action take more than an hour to finish, whoever volunteered to take the action needs to carry it with them into sprint planning. It can then be included on the task board for the next sprint.

RETROSPECTIVE CHALLENGES

As already mentioned, your retrospective should be at least somewhat entertaining and engaging. A team unfamiliar with agile might have some difficulties at joining in a retrospective, so they should be designed to include these people. Difficulties of this sort arise for a variety of reasons, but many of them are related to fear:

- Fear of blame. Perhaps a story remained incomplete after the sprint, or maybe something occurred for a team member that slowed down the whole team. As such, there might be an understandable fear of blame or reprisal during a retrospective. Avoid this by being sure to spotlight your team's attention on collective aims and continual overall improvement. Discuss things with potentially problematic team members before a retrospective to gain insight on their problems.

- Fear of wasting time. A team member might consider a retrospective redundant, and you will need to determine the reason for this attitude. Perhaps they did not see any improvement from actions (or lack thereof) applied after a prior retrospective. Perhaps they are overwhelmed by other projects and hesitant to take the time to look back.

- Fear of speaking up. Some team members may not have the confidence or certainty to be vocal during a retrospective. You can take care of this problem with a transferable speaking "token" passed around to each team member. You should make it acceptable to decline the token if a team member does not have anything to add. You should take the time to speak one-on-one with any team member who appears uncommonly silent over a course of several sprints, though.

Never highlight and single out quiet team members in a retrospective. By picking them out of the team, you might castigate them and cause them to disengage further. It might even give the impression of blame. Rather than talk in the retrospective, take them aside to discuss any issues in private.

If this is a concern, another technique is a silent retrospective, performed using sticky notes and affinity mapping. This has the benefit of not putting any individuals on the spot in front of the group, since that is definitely not the objective of this activity.

THE SPRINT RETRO-SPECTIVE IN A NUTSHELL

Sprint retrospectives take place after a sprint review, which means teams can pause and consider the sprint as a whole, as well as the review itself.

Let’s recap:

- A sprint is not over until the team holds a retrospective, even though it takes place on the last day of the sprint.
- By holding a retrospective at the close of each sprint, you offer the team a chance to advance progress on project delivery.
- A retrospective is the final inspection point in a series of in-sprint inspections, offering the ideal chance to recognize ways of adjusting and advancing agile methods.
- At the very least, the Product Owner, Scrum Master, and development team need to be present for a retrospective. Support teams and other stakeholders should definitely be invited.
- For a thirty-day sprint, you should typically expect a retrospective to take about four hours. This can be adjusted to the requirements of the team and the project itself.
- To begin with, it is normally the Scrum Master who regulates a retrospective, though other team members should be able to facilitate if they so desire.
- If your team is being supported by an Agile Coach, that person should also be involved.
- Use games to make a retrospective more productive, efficient, and entertaining!

10. METRICS: STOP MEASURING THE WRONG THINGS

“Statistics are no substitute for judgment”. ~ Henry Clay

MEASURING UP

It is much more difficult to measure non-performance than performance. Performance stands out like a ton of diamonds.

Non-performance can almost always be explained away.

~ Harold S. Geneen

Previous information has described how to become agile, but it is now time to discuss metrics so that you can continually evaluate the impact of the agile transformation on your business.

Metrics are essential for every project, regardless of how you deliver them. They let you gauge and communicate your team's development, a key part of the agile process. Metrics will also highlight which parts of your process need improvement.

At this point in your agile journey, we are going to address how one produces useful metrics that allow you to observe the advances in your agile project.

As your project advances, the metrics involved will naturally change. You are advised against beginning your project unless your team is aware of the metrics your stakeholders demand.

Be Prepared

Once your agile team has begun their opening sprint, it is vital to understand which metrics need to be recorded, why they are being recorded, and who receives them.

Coming to grips with the data and the requirements of your stakeholders lets you clearly and openly give feedback on your team's condition, particular to senior management. The reports generated by the Scrum Master or project manager might also need to be altered to meet the demands of a variety of stakeholders, depending on the type of project.

This is where using automated software tools to help track and communicate vital metrics for appropriate stakeholders can be invaluable.

Metrics isn't a case where one solution is appropriate for every scenario. The way you use metrics may need to be adjusted from one project to the next. The list of metrics covered here is in no way exhaustive.

Hard Metrics

Projects typically cover two different types of metrics, namely hard or soft. Hard metrics cover the mechanics of agile projects and need to be reported in Burn Down Charts, Burn Up Charts, defect reports, and build failures.

THE BURN DOWN CHART

Measure three times before you cut once.

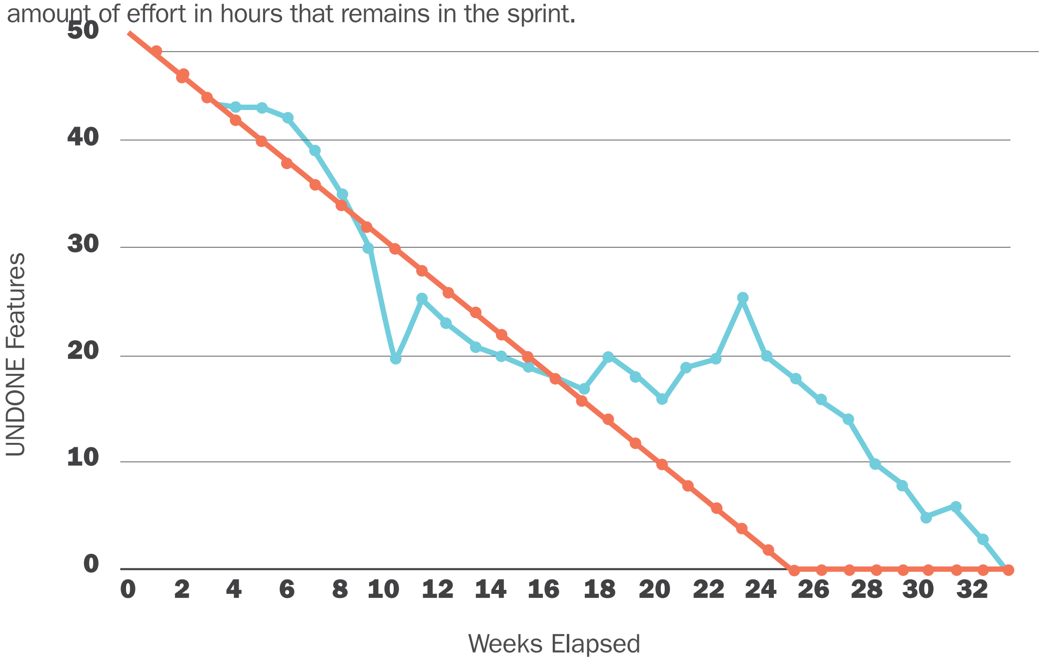
~ Proverb

Some would say that the most helpful report for charting the advances of an agile project would be the Burn Down Chart, which records the work (ideal days, points, hours) that still needs to be done as compared with the sprints in a project or the days in a sprint.

As such, Burn Downs can be observed and used to measure both project and sprint.

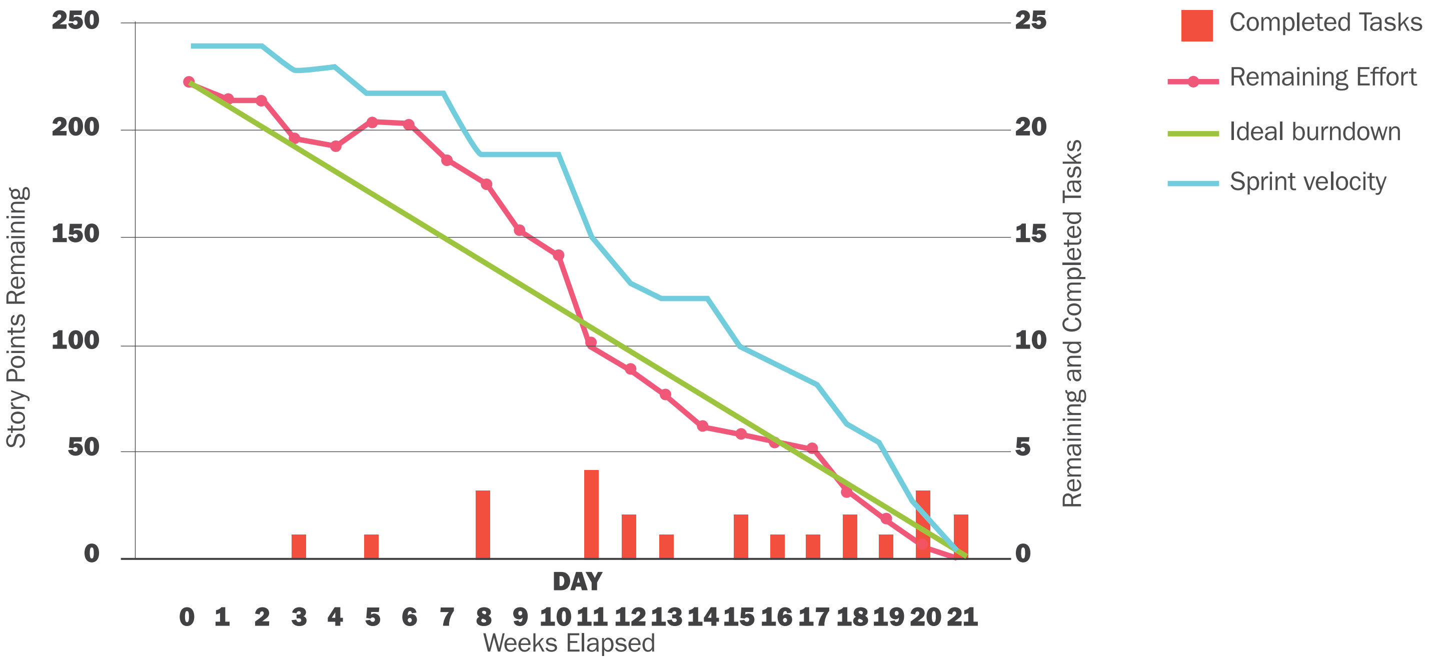
In terms of a project, you use the X-axis to represent the number of sprints and the Y-axis to chart the work remaining, whether this is represented in terms of ideal days, story points: basically, whatever system your team has settled on.

In terms of a sprint, your X-axis shows how many days are in a sprint, while your Y-axis portrays the amount of effort in hours that remains in the sprint.



Software Velocity Burndown Chart

EXAMPLE: Assume a particular team's velocity is approximately 41 story points per sprint. Based on Figure 9 below, if we presume that everything is equal and that the number of points does not increase or decrease, we can expect this team to complete all 260 story points in seven sprints or so.



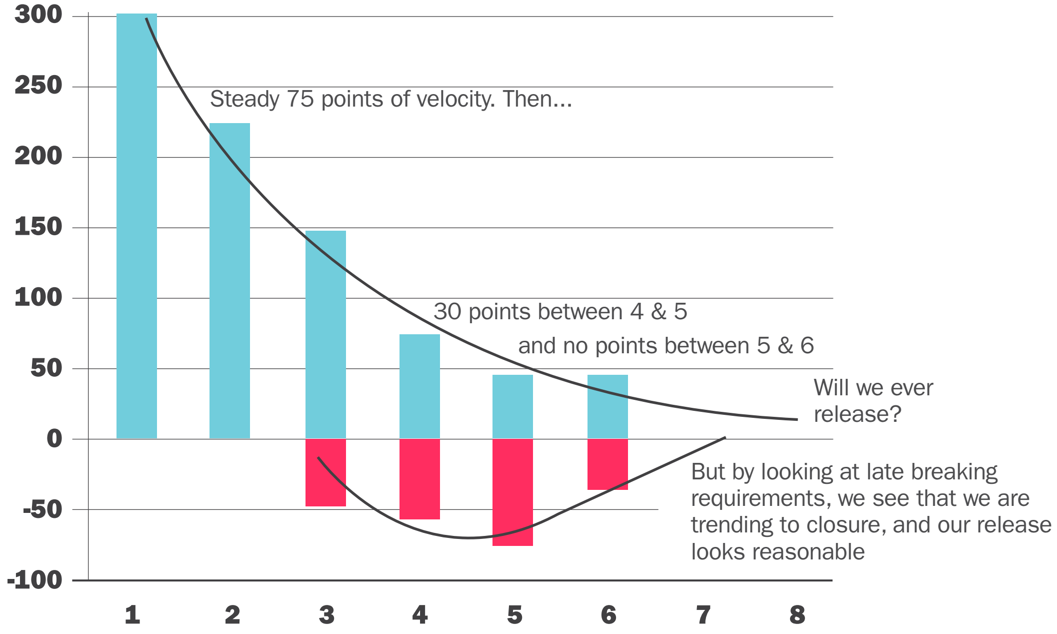
Sample Burndown Chart

As you can see in the samples, the number of hours remaining may not trend smoothly to zero, which is just how Burn Downs operate. Work can be added and removed, thus the trend line will mount or dip on its route to the bottom. Take on your team's feedback in the daily Scrum or stand-up to gauge this progress, always making certain to trust what your team tells you.

RELEASE BURN DOWN CHARTS

One weakness of Burn Down Charts is that they do not always show how much work has been completed in a sprint or how the amount of work in a project has altered. A casual observer, looking solely at the Burn Down chart, might not grasp how much work has been added, discarded, or completed in the project.

Stakeholders who cannot understand this sort of information may become needlessly worried, as explained in the following example.



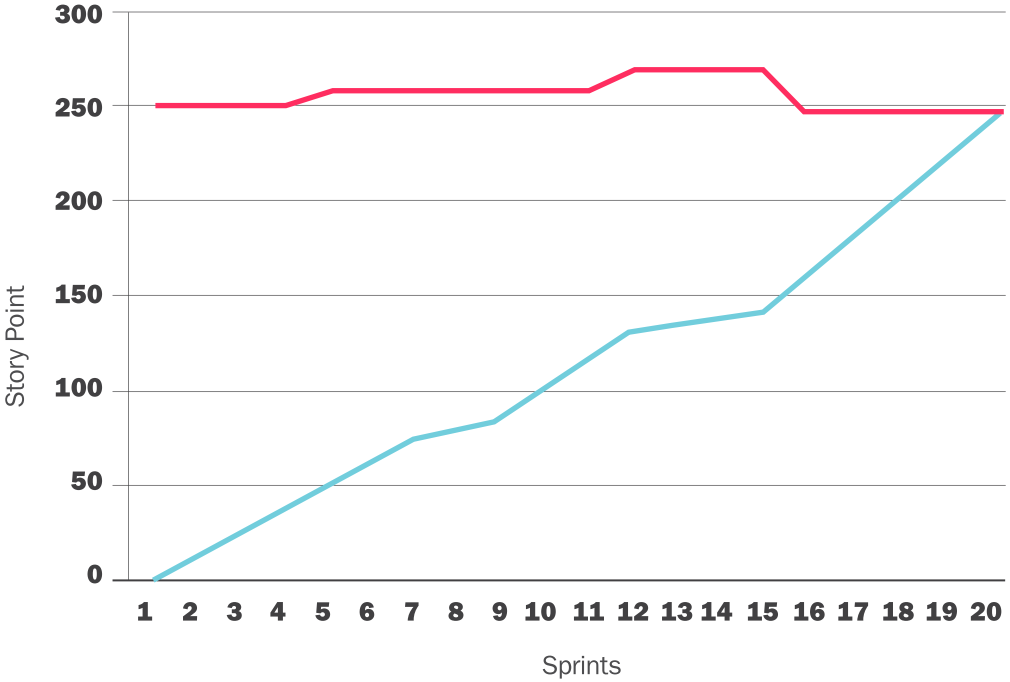
EXAMPLE: In the Burn Down Chart above, when looking at Sprints 4, 5, and 6, it seems as if the team only did about 30 points of work in Sprint 4, as measured between 4 and 5. In Sprint 5, as measured between 5 and 6, it seems that no work was completed at all with work still being added. This back and forth trending keeps up throughout all the project's remaining sprints. Should project stakeholders only glimpse the project through this Burn Down Chart without any other reports, it might cause undue concern about the project slowing without any reason as to why.

Should stakeholders want to know when a team will finish all work in the project backlog before you have even determined a team's velocity, it is up to whoever reports the status to set realistic expectations by getting management to comprehend the significance of empirically determining future projections. Highlight the importance of projecting the number of story points that a team can deliver in upcoming sprints by observing what has been delivered in prior ones. If absolutely pushed, get together with your team and offer best/worst-case scenarios but do not commit to any number of sprints or a precise date, as described earlier.

THE BURN UP CHART

Burn Up Charts normally depict story points on a project level. With a Burn Up Chart, your X-axis portrays the sprint and your Y-axis the story points that have been completed and the overall story point total in the project.

Generally with story points, Burn Ups are particularly good at demonstrating the amount of work as a whole for the project and the amount of work completed in each sprint, as well as ascertaining team velocity, which can then help you estimate, based on your current backlog, the number of sprints remaining until the project is finished.



EXAMPLE: In the Burn Up Chart above, the line for “Work Complete” shows the total number of story points in the project. Whenever user stories are included or discarded in the project, they are incorporated into the Burn Up Chart. The line for “Work Complete” depicts the user stories that have been completed.

A project is over when the two lines connect. The disadvantage to a Burn Up Chart is that the total number of story points that still need to be completed is not instantly obvious. To determine the number of story points to completion, you have to subtract the latest number on the “Work Complete” line from the latest number on the “Scope” line.

So, if the team takes 50 from 250 at the end of Sprint 4, this leaves them with 200 story points until project completion.

Though the number of story points left in the project can be figured out using a Burn Up, you have to keep your audience in mind when it comes to reporting this data. More often than not managers will need the numbers shown to them in simpler form. As a result, it is usually beneficial to offer both Burn Up and Burn Down Charts so as to show the whole picture for the project. If you can aggregate this information into one easily comprehensible format, such as “X points to completion,” then all the better.

THE DEFECT REPORT

So far we have discussed calculating both project and sprint progress, which helps you determine the speed of your deliveries. However, Burn Up and Burn Down Charts do not give you any idea of the quality of the product that you are delivering.

As previously mentioned, one of the major advantages to agile is that it helps in the creation of better quality products.

With this in mind, stakeholders will want to know how many defects a current sprint might have discovered, as well as the types of defect when compared with previous sprints. Once again, software-testing tools can be a real help with producing defect reports as any issue arises.

EXAMPLE: Did you notice fewer defects in your agile project this year, when compared with a waterfall-based project last year? If you examine this data in detail, you can classify defect comparisons into different categories, such as Critical, Average, or Minor, etc. You can also compare across a variety of testing types, like Unit, Integration, and User Acceptance.

When your company is changing from waterfall to agile, should you need to demonstrate the positive changes that have occurred, it would be helpful to show the total defect rate for a product over a series of projects.

BUILD FAILURES

When you use build failures as a metric, a team can learn the number of builds broken within a sprint and across a number of sprints. Build failures help you envision code quality when going into a build and aid a team in their decisions when it comes to their engineering practices.

When this is combined with the previously mentioned continuous integration, then understanding build failures lets a team determine exactly who is submitting bug-ridden code to the repository.

Team and Human Performance Metrics

Having understood hard metrics and how they deal with the mechanics of agile projects, we can now tackle Human Performance Metrics, which address team behaviors as well as emergent collaboration and technical execution skills of individual team members. More often than not Human Performance Metrics are simply not collected and reviewed, which is unfortunate because these metrics provide a compelling and predictive view of the future. Teams that are improving in their agile collaboration and engineering skills are teams that will produce at greater levels of predictability and quality on future sprints. Measuring these metrics is a strategic way of thinking about how to invest in your team's future capabilities.

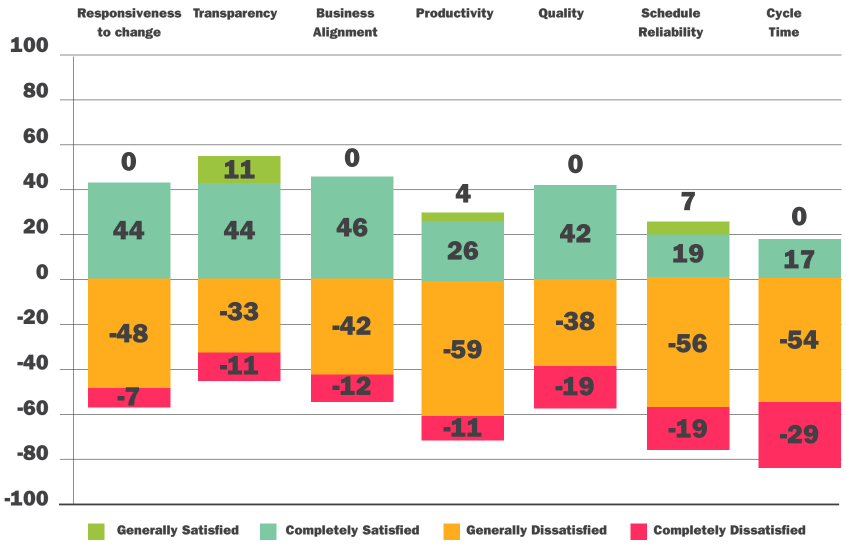
Think of this as an early warning health management system allowing companies to intervene and invest in important team and individual improvements BEFORE a team experiences the stress and disappointment of poor performance. There are models and emerging tools for assisting with these Human Performance Metrics.

CUSTOMER SATISFACTION

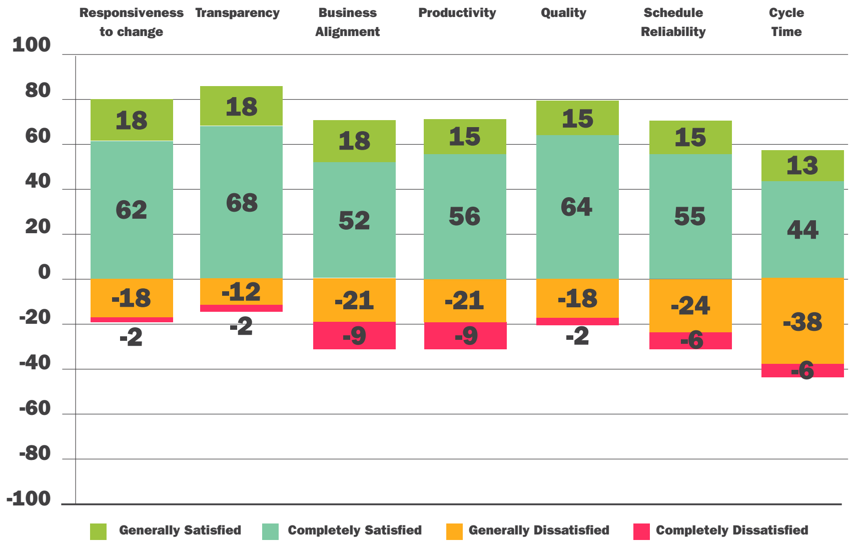
The ultimate metric of any product is whether or not the customer is satisfied with the way the product suits their needs. Customer satisfaction is so critical that the Product Owner, as a representative of the customer, has daily access to the development team over the course of the whole project. Though they provide feedback on customer satisfaction, it is not just the Product Owner but the team itself that will need to collaborate with end users to evaluate how content they are with the overall product. The ideal time for this is the Sprint Review.

But what if your customer stakeholders include more than a simple Product Owner? In this instance, we recommend you consider adopting the Net Promoter Score (NPS) framework, described in detail at http://www.netpromoterscore.com/. It's a simple and industry-recognized framework for quickly understanding the trend and health of your customer's satisfaction. Many agile teams have found NPS a simple, yet powerful way to obtain feedback on the agile execution and collaboration improvements they're making. It's equally useful in evaluating the level of satisfaction your users and customers have with the finished software and features you're delivering.

Net Promoter Scores - Baseline



Net Promoter Scores - 6 Months



METRICS THAT DON'T MATTER

Not all metrics are created equal. Some are critical for evaluating the behavior of a team or status of a project, but others provide irrelevant or, even worse, misleading information. As your company goes from waterfall to agile practices, or even just tries to improve on their existing agile practices, some old metrics will need to be discarded.

Discard Percent Complete

Percent complete shows the percentage of work that has been completed overall for a task or feature. If a task has been estimated at eight hours and there are four hours until completion, it would be presumed that the task was halfway, or 50 percent, done. Surely this would be useful information to track, right?

No. This information might be useful if the percent complete is identical to the percent that remains to be completed, but that's rarely the case in industries with an emphasis on creativity and innovation, such as the software industry. Actually, the final 20 percent is typically the hardest or most arduous section!

In an agile project, the system is binary: something is either done or it isn't. There are only two percentages: zero and 100. If someone were to claim that a task or feature was 80 percent complete, this would equate to zero in agile. It simply is not done. By trying to gather percent complete reports, you put unnecessary pressure on a team and draw attention from the real concern, which is getting the job done.

Management will probably want percent complete reports simply because this method is traditional, and that is what they are used to. If this is the case, you will need to educate management about the advantage of not keeping tabs on percent complete.

The information that management really needs varies. Ask them. You can tell them which stories are really done. They have the benefit of the sprint review to see them in action as well as the working software. You can also extrapolate based on empirical evidence of past progress using the Burn Up Chart.

TRACKING ACTUAL FEATURE OR TASK HOURS

Organizations usually have a time-tracking tool for members when they are participating in multiple projects. In an agile project, tracking real hours at a project level is common sense when team members are taking part in multiple projects under multiple cost centers. This applies to internal as well as external projects.

Then again, for internal projects, keeping track of actual hours at a feature or task level is pointless. Many project and people managers operate under the misapprehension that by forcing a team to track actual hours at a task or feature level, they will become better at estimation. In software development, though, this kind of accuracy is not exactly possible, since two features are seldom ever alike.

Agile team members will also take part in multiple tasks, such as e-mails and meetings, throughout their working day. In recording actual hours, these are more like estimated actual hours. Just like percent complete, keeping track of actual hours on feature and task levels puts unnecessary pressure on a team.

Should your organization be developing an application for an external customer or if an external vendor is in the process of developing an application for your organization, then details concerning the tracking of actual hours will depend on the precise nature of the contract.

METRICS IN A NUTSHELL

Metrics are vital to every project, even if it isn't agile. We've covered a lot here, so let's recap:

- Even before you begin an agile project, you should determine which metrics to gather and who they are for.
- Burn Down Charts keep track of information on both the sprint level and the project level.
- Burn Down Charts are useful for determining team velocity and projecting how many sprints are needed to finish a project.
- Burn Up Charts are normally shown on a project level and use story points.
- Burn Up Charts depict the work that has been finished and any changes to the workload.
- A defect report shows the total number of defects over the project.
- Defect reports are helpful in determining increases or decreases in defects in terms of previous projects, releases, or sprints.
- Customer satisfaction and team morale are vital metrics in tracking agile projects.
- Percent complete and actual feature/task hours are irrelevant metrics and should be discarded.

11. IS SCRUM ENOUGH?

"Man cannot discover new oceans unless he has the courage to lose sight of the shore." ~ André Gide

AGILE

COMPLEMENTARY AND ALTERNATIVE AGILE METHODS

Each company will have its own slant on what it means to be agile, which has produced a variety of complimentary methods. Nevertheless, all draw their inspiration from the Agile Manifesto.

The agile processes we have covered here are primarily drawn from Scrum, but many other compelling engineering and alternative agile project management frameworks are available today that should be considered.

In fact, Scrum will likely meet with limited success if you do not consider integrating and adopting many of the following methods:

Complementary to Scrum:

Engineering Mastery: XP / Extreme Programming

- Test-Driven Development
- Pair Programming
- Continuous Integration
- Refactoring

Cycle-Time Optimization: Kanban

- Work In Progress (WIP) Limits
- Visual Controls
- Cumulative Flow Diagrams

Alternatives to Scrum:

Feature-Driven Development

- Building a features list
- Planning by feature
- Designing by feature
- Building by feature

Dynamic Systems Development Method

- Frequent Releases
- Iterative Development, Driven by User Feedback
- All Changes Must Be Reversible
- Requirements are Initially Defined at a High Level
- Fitness for Business Purpose is the Goal
- Integrated Testing
- The Pareto Principle (also called the 80-20 Rule)