Sprint Planning
Backlog Grooming
Scrum of scrums

Glossary of Agile Terms

There are as many versions of agile as there are agile teams, with each team adopting different rituals, terminology and tools. This glossary lists many of the terms you may hear when working with agile teams. While this is by no means exhaustive, these will be helpful in translating the most common jargon.

In some cases, there are official definitions for these terms and also some more colloquial usages. We've tried to call those out where we can.

5 Whys

The **5 Whys** is a technique for finding the root cause of problems. When something goes wrong, the team asks what caused or allowed the bad thing to happen. They then ask why that happened. It often takes more than five iterations to get to the root cause of a problem, which makes the "5" a bit arbitrary, but the idea is that you keep asking why until you understand what you need to fix.

It's commonly used on agile teams, especially during post-mortems, where teams have to figure out what went wrong collaboratively.

Learn more about the 5 Whys technique here.

Acceptance Criteria

This is the list of requirements a user story needs to have in order to be considered "done" or ready to ship to customers. **Acceptance criteria** are often created by product owners, product managers or scrum masters, depending on the composition of the team. Acceptance criteria can include technical requirements, user requirements or business requirements.

Some teams have standard formats for acceptance criteria in order to make it easier for engineers to know exactly what needs to happen before they can declare a story done.

There are some examples of formats and criteria here.

Acceptance Testing

Acceptance testing is the process by which somebody on the team verifies that a story meets the acceptance criteria. This is often conducted by product owners, product managers or scrum masters, but could be done by quality assurance (if they exist on the team) or designers who know how a feature is supposed to work. It's a final check that something really does what the engineer says it does before code gets released to end users.

Agile Manifesto

The **Agile Manifesto** is a document published in 2001 by a bunch of engineers who were working with various lightweight software methodologies. It defines an overall view of software development they named Agile.

Read the Manifesto.

Anti-Pattern

Any commonly occurring pattern or behavior that is generally negative. In the context of the course, we talk about agile **anti-patterns**, which are things that often happen on teams that say they're agile that don't create a positive or even a particularly agile environment. For example, never iterating is an unfortunate anti-pattern that we see on a lot of teams that claim to be agile.

Backlog

The **backlog** should consist of user stories and epics that are ready (or at least almost ready) for engineering to start working on. It should be prioritized, and it should be limited to things that we are fairly certain should be done soon. The **ice box** is often used to refer to things that might someday happen or that are simply at the idea stage and haven't been validated yet. They are typically managed by scrum masters, product owners, or product managers, depending on how your team is set up.

If you noticed there are a lot of "shoulds" in that definition, you're correct. In reality, backlogs are often an unprioritized, unvalidated mess of feature ideas, bugs and tech debt.

Backlog Grooming

Backlog grooming is the process of going through the backlog and sometimes the ice box to prioritize the tasks and make sure that stories are ready to be worked on. On good teams, there are backlog grooming meetings including representatives from product, engineering and design where everybody can have input into priorities.

Big Design Up Front (BDUF)

Big design up front (**BDUF**) is often used to describe a sort of waterfall process where the entire research and design process of a product or large feature is done before engineering is involved. It is typical in many agency or siloed design team projects.

Blocker

A **blocker** is anything that is keeping somebody from performing a specific task. For example, design can be a blocker on an engineering task if the engineers are waiting for final assets from design in order to implement them. Product can be a blocker on a task if the team is waiting for a final decision on a business matter. Blockers are often discussed during stand-up meetings so that everybody knows if somebody is stuck on something.

Continuous Integration/Continuous Deployment (CI/CD)

These specifically relate to how new code gets pushed out into production. Once upon a time, engineers would work on a new feature or a new version of software until it was done, and then they would release all the code to users at once. In the days of CD-ROMs, that would be in the form of a new box of software. For many pieces of software, you'll get a new version pushed out occasionally.

With **continuous deployment**, code is pushed out all the time, in some cases multiple times a day. This can mean that there are small (or large) changes to the product quite a bit, although not all code changes result in large changes for the end user. Sometimes it's something like speeding up a process or changing something on the back end to make it more stable. If it is user facing, sometimes it's something like fixing a bug.

Continuous integration is similar, except that the code doesn't always get pushed all the way out to the users. But it is shared internally on a regular basis, which cuts down on what are called "merge conflicts" where two engineers make changes that affect each other.

As a designer or researcher, you don't need to know a lot about the mechanics of this, but it definitely can affect how your team thinks about changing the user interface. They may be in favor of making lots of small changes to the interface on a regular basis rather than saving them up and making one large change. It can affect the way you design in small batches, so it's a good thing to discuss with your engineers.

Continuous Discovery

Continuous discovery is a method used by product teams to constantly get feedback and insights from users rather than concentrating all of their discovery work into one set time period.

Read more about it on Teresa Torres's website.

Cross-Functional Team

Cross-functional teams are made up of people with different skills and specialties. A cross-functional product team should have enough of each sort of job role to produce the product they're making, so that they're not relying on people in other silos to deliver work to them. A typical software product team would generally have engineers, one or more designers, one or more researchers, and a product manager or owner.

Some of the roles are often combined. For example, a product owner might lead the research rather than having a dedicated researcher. Some teams might have other specialists like a data analyst or quality assurance specialist. Often people will play multiple roles. For example, an engineer might be a tech lead and/or a scrum master.

Demos

Demos (short for "demonstrations") are done during a meeting at the end of the sprint. They are generally times for engineers (or sometimes other members of the team) to show what they finished during the sprint. They are an excellent time to make sure that what actually got built is what everybody thought would get built. They are also a celebration of the work everybody did.

Epics

Epics are larger pieces of work — often whole features — that are broken down into user stories or tasks that can be worked on independently. Epics often take multiple sprints to complete. Stories are grouped into epics so work that needs to be divided into many small pieces can still be tracked and the team can know when it's done.

Often epics aren't released to users until certain milestones of completeness have been met. For example, you wouldn't want to release a job board that didn't let people see the posted jobs.

Estimation

Estimation is the process of saying how long you think something will take. On scrum teams, estimation of user stories is often done in the form of “story points” so that we can roughly estimate the size of any given story and decide how much work to do in a sprint.

Estimation of stories should be done by whoever will be executing the work. In reality, this can vary, and on some teams, the scrum master or product owner or tech lead will provide the estimation. This is generally not a great practice, though.

Extreme Programming (XP)

Extreme programming (**XP**) is a software programming framework developed by Kent Beck that is one of the precursors to the Agile Manifesto. Many of the principles in the Manifesto come from the ideas of extreme programming.

Feature Flags

Feature flags are pieces of code that allow the team to turn certain functionality in a product on or off all at once. Often, in a continuous deployment or continuous integration environment, engineers will want to push code into production before a feature is complete. A feature flag allows them to have all the code pushed and ready to go, but keeps the feature “off” in production until somebody flips the switch and turns it on. Good feature flagging systems will also allow you to make some features available only to specific groups of people, which is extremely useful for testing real code with beta users.

Ice box

The **ice box** is where product owners throw all the “great ideas” for features that will probably never actually be built. It is typically the longest list in the company.

Information Radiators

An **information radiator** is any artifact (usually a physical one) that is placed in a communal team area to provide important information to the team. For example, a poster with persona information or a screen showing real-time metrics would both be considered information radiators.

Jira

Jira is a project management tool made by Atlassian that is used by a lot of agile teams. You will often hear people talk about “Jira tickets” rather than user stories, because that’s where a lot of teams organize their sprints, backlogs and ice boxes.

Kanban

Kanban is a project management process designed to improve work and efficiency. It’s one of the precursors to the Agile Manifesto, and many of the ideas we think of as agile come from Kanban.

Kanban Board

A **kanban board** is used to visualize and track progress of tasks over time. It’s one of the most commonly used tools in agile teams.

Pairing

Pairing (or sometimes mobbing, if there are more than two people) refers to two people working together on implementing the same solution. It was originally used to describe pair programming, where two engineers would work together to write code, rather than every engineer working separately. Now we often refer to designers pairing with engineers. This generally means that the designer sits with the engineer while the code is being written to help figure things out on the fly. It can be especially effective for user interface changes, since it can prevent having to write long, time-consuming specifications for every UI element.

Planning Poker

This is often used by engineers to do group estimations of how long user stories will take. It is typically done at the planning meeting or backlog grooming. It’s only one of several ways that people estimate stories, but it’s a fairly common one.

Post-Mortems

These are generally held after something bad happens. For example, if a bad user-facing bug makes it into production or there is some sort of data breach, the team would want to hold a post-mortem to figure out:

01 Why it happened.
02 How to fix the damage.
03 How to prevent this problem or similar problems from happening in the future.

Post-mortems of this sort should always end with specific changes to process or code that are meant to improve the system. On good teams, post-mortems are "blameless" — which means the goal shouldn't be to place blame on an individual but to identify areas of the system that need to be improved to avoid problems in the future.

Teams will frequently use the 5 Whys or a similar tool to find the root cause of a systemic problem instead of simply fixing the symptoms of the problem.

Product or Marketing Requirements Document (PRD/MRD)

These are documents that are much more common in waterfall environments. They include all the business and user requirements for a product or feature, which are typically all specified early in the development process. They are then generally handed off to either design or straight to engineering to complete their part of the process.

Agile teams do not tend to produce these documents, because they're basically the antithesis of agility. However, you may still see them on teams that claim they're agile if they're only using "agile" to make the engineering team work faster.

Product Owner

The **product owner** is often described as the liaison between customers and engineers. It is sometimes used synonymously with the term product manager. The actual job description of the product owner varies widely between teams.

Read more about the difference between product owners & product managers on Teresa Torres's blog.

Refactoring

Refactoring is a term used mostly by engineers to describe changing the underlying code without changing the behavior of the program. We also sometimes use it to describe small redesigns of pieces of the user interface in order to allow for new functionality.

Release

A software **release** describes when new or updated code is made available to users. In a continuous deployment environment, code is constantly being released (often called “pushed”) to users, sometimes multiple times a day. In a waterfall environment, code might only be released once or twice a year.

Retros

Retros (short for retrospectives) are meetings held by the team at the end of each sprint. The goal is to reflect on how the sprint went and find ways to improve the process. There are several models for holding good retros, but generally speaking, the team should think about what went well and what went poorly and make decisions about any process changes they want to make in the future to make things run more smoothly.

Scrum

Scrum is a project management framework that is frequently used on agile teams. A lot of the things we associate with agile methodologies (sprints, scrum meetings, planning meetings, etc.) actually come from Scrum. There are several organizations that offer certification in various different Scrum-related skills and roles.

Scrum Master

The **scrum master** is the person on the team who is responsible for improving processes, ensuring agility and making improvements to how the team works. It should be a role on the team, not a job title, although this varies widely depending on the company.

Scrum of Scrums

This describes organizations that have a large number of scrum teams with a single team of representatives from each scrum team who meet to keep everybody organized.

Self-Organizing

The idea behind **self-organizing** teams is that they decide how to accomplish their tasks and goals. Instead of having tasks doled out to team members by a tech lead or product owner, each member of the team selects the tasks they are most suited to working on.

Semi-Autonomous Teams

Autonomous teams decide what to work on based on higher-level strategy goals. For example, instead of being assigned to build a feature, a team would be asked to hit a specific metrics goal and could decide for themselves which feature was the most likely to do that.

In this course, we often refer to these teams as “**semi-autonomous teams**” because in the real world, only very small teams are truly autonomous. In most cases, scrum teams still need to coordinate with other teams within the company to make sure nobody is duplicating work or building a competing feature.

Sprint

A **sprint** is a period of time (generally between 1 and 4 weeks) during which a team commits to accomplishing a specific set of work.

Stand-Ups

Stand-ups or **stand-up meetings** (or scrum meetings) are short status updates that generally happen every day, or at least every day that doesn’t have another major scrum meeting. The format of a stand-up is for everybody on the team to quickly explain what they did yesterday, what they’re doing today, and what is blocking them from getting their work done. The goal is to make sure everybody on the team stays informed and can help unblock their teammates if necessary.

Story Points

Story points are often used in estimation. Instead of saying something will take 4 hours or a day or some other increment of time, engineers (or designers when asked to estimate their design time) will assign points to a user story. Higher point values mean larger tasks. Generally, teams that use story points will aim for a particular point value every sprint. For example, a team may find over time that they can accommodate roughly 30 points in a sprint, so if they’re over that, they may put off a story until the next sprint. Some teams also use the points to track velocity (how fast they finish stories) over time. If you’re thinking that this probably just leads people to assign larger and larger point values to stories to make it seem like they’re working faster, you’d be correct.

Tasks

Tasks are units of work that need to get done. Tasks are often tracked on a kanban board. The term task is sometimes used synonymously with user story or story, but not always!

Tech Debt

Tech debt (and its friend UX debt) represents engineering decisions that the team made in the past that need to be fixed. Sometimes we intentionally take on tech debt in order to get something out quickly for various reasons, but quick hacks need to be fixed, and if we do it too often without improving the code architecture, it can get much harder to make changes in the future. It can even end up ruining the product.

UX debt is similar. If you've ever worked on a product that became wildly inconsistent or very hard to use because years of extra features had been added on without anyone ever going back and fixing the old stuff, you've experienced UX debt!

User Stories

User stories are supposed to represent a description of a feature (or part of a feature) from the user perspective. They will often be expressed as:

As a **<who>** I want **<what>** so that **<why>**.

Originally, the goal was to write them in such a way that the engineer could then find the best possible way to implement them. However, for teams that have researchers and designers, it can be tricky to figure out when these should be written and by whom. Often, these are the kind of stories that come out of a research and design process and have complex interface designs that go along with them. However, on some teams, designers are assigned user stories in order to build the UI to make them happen.

The treatment of user stories varies widely from team to team, and you'll have to understand how your team uses them in order to fit them into your design process.

Velocity

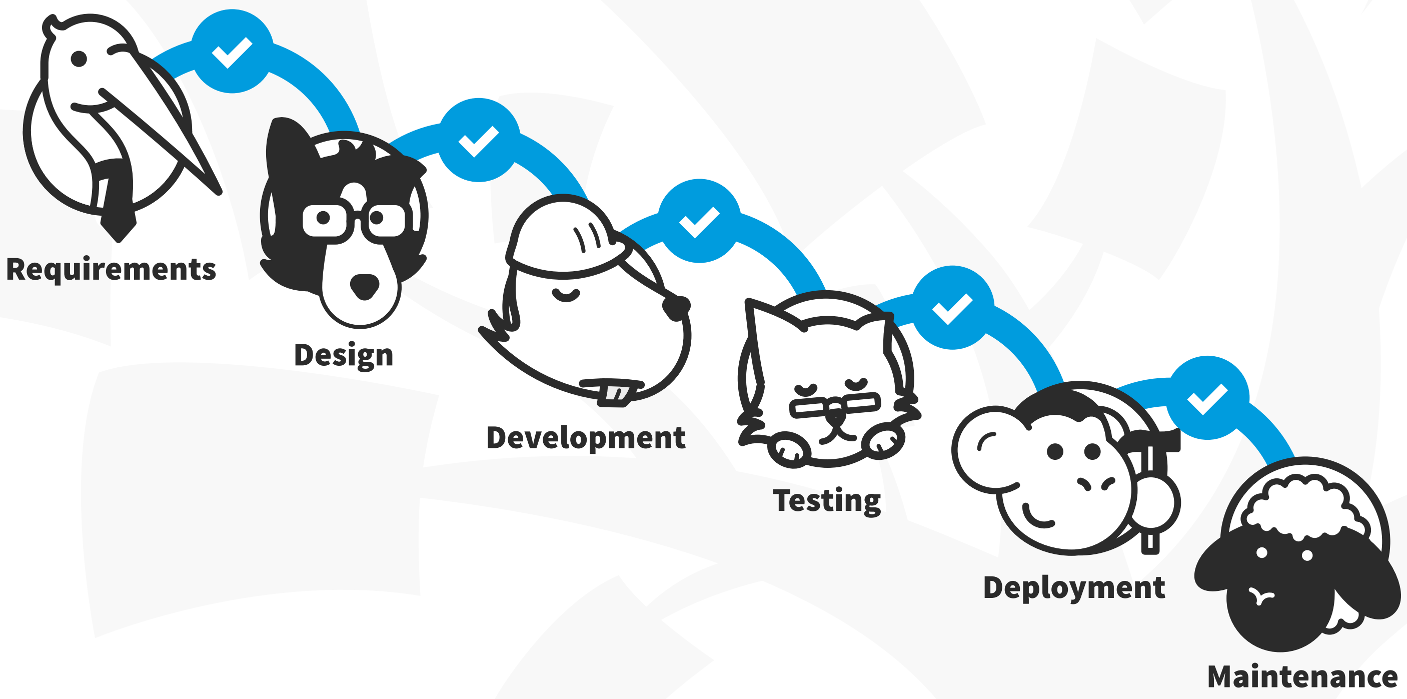
Velocity is the speed with which a team delivers user stories as code. It's generally expressed in story points.

Waterfall

Waterfall was the most popular project management style for software (and other) products before agile. It is still widely used in many companies and industries. In waterfall, each team completes their portion of the work which is then verified before moving onto the next phase of the product. This means that all the product requirements are written before the product is designed. The entire design should be done before being handed off to engineering. The engineering should be more or less finished before we do the final verification of the code.

Agile is a reaction to the failings of the waterfall process, in a lot of ways. On agile teams, we acknowledge that we can't know every user need and every technical or design constraint upfront, and we need to be able to react to user needs and changes in the environment as we build products.

Waterfall Process



Do You Want to Learn More?

Methods of using this template are taught in our online course **Agile Methods for UX Design**. Make full use of this template and learn more about working on agile teams by signing up for it today.

Agile Methods for UX Design

Intermediate Course

Agile, in one form or another, has taken over the software development world and is poised to move in to almost every other industry. The problem is that a lot of teams and organizations that call themselves "agile" don't seem to have much in common with each other. This can be extremely confusing to a new team member, especially if you've previously worked on an "agile" team that had an entirely different definition of "agility!"

The **Agile Methods for UX Design** course aims to show you what true agility is and how closely agile methodologies can map to design. You will learn both the theory, and the real-world implementation of agile, its different flavors, and how you can work with different versions of agile teams.

This is an intermediate-level course for people who already know how to design or research (or who want to work with designers and researchers) and want to learn how to operate better within a specific environment.

You earn a verifiable and industry-trusted Course Certificate once you've completed the course. You can highlight it on your resume, your LinkedIn profile or your website.

Learn more about this course >

Start Learning Today:
interaction-design.org

How to Advance Your Career With Our Online Courses

Take Online Courses by Industry Experts.

Lessons are self-paced so you'll never be late for class or miss a deadline.

Get a Course Certificate.

Your answers are graded by experts, not machines. Get an industry-recognized Course Certificate to prove your skills.

Advance Your Career.

Use your new skills in your existing job or to get a new job in UX design. Get help from our community.

See all our courses >

About the Interaction Design Foundation

With over 100,000 alumni, the Interaction Design Foundation is the **biggest design school globally**. Industry leaders such as IBM and Adobe train their teams with our courses, and universities such as MIT and the University of Cambridge include our courses in their curricula. Our online courses are taught by industry experts and cover the entire spectrum of UX design from beginner to advanced. We give you industry-recognized course certificates to advance your career. Since 2002, we've put together the world's biggest and most **authoritative library** of open-source UX Design literature created by such noted authors as Don Norman and Clayton Christensen.

INTERACTION DESIGN FOUNDATION

Start Learning Today:

interaction-design.org