Scrum tutorial

**What is Agile?**

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

Agile can be implemented using various frameworks (like [scrum](https://www.atlassian.com/agile/scrum) and [kanban](https://www.atlassian.com/agile/kanban)) to deliver software. Scrum teams use sprints to guide development, and kanban teams often work without fixed work intervals.

**WHAT IS SCRUM?**

[Scrum](https://www.atlassian.com/agile/scrum) is one of the most popular frameworks for implementing agile. With scrum, the product is built in a series of fixed-length iterations called sprints that give teams a framework for shipping on a regular cadence.

**WHAT IS KANBAN?**

Kanban is similar to Scrum, in that it helps teams release software early and often. However, Kanban gives more flexibility in terms of planning and execution. Instead of working in time-based sprints, work is continuously delivered in Kanban, and your team pulls single pieces of work from the backlog, and then moves them to done.

**WHAT ARE USER STORIES?**

[User stories](https://www.atlassian.com/agile/project-management/user-stories) are used to describe work items in a non-technical language and from a user's perspective. As a {type of user}, I want {goal} so that I {receive benefit}.

**WHAT IS A SPRINT?**

In Scrum, teams forecast to complete a set of user stories or other work items during a fixed time duration, known as a sprint. Generally speaking, sprints are one, two, or four weeks long. It's up to the team to determine the length of a sprint — we recommend starting with two weeks. That's long enough to get something accomplished, but not so long that the team isn't getting regular feedback. Once a sprint cadence is determined, the team perpetually operates on that cadence. Fixed length sprints reinforce estimation skills and predict the future velocity for the team as they work through the backlog.

**WHAT IS A SPRINT GOAL?**

When creating a sprint, the product owner usually identifies a sprint goal. This provides a theme for the work to be completed in the sprint. A sprint goal also provides some flexibility in the number of stories that are completed in a sprint. A sprint is considered a success if the sprint goal is achieved.

**WHAT IS AGILE ESTIMATION?**

Traditional software teams give estimates in a time format: days, weeks, months.  
Many agile teams, however, have transitioned to story points. Story points rate the relative effort of work, often in a Fibonacci-like format: 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100.

**WHAT IS A BURNDOWN CHART AND HOW DO YOU READ IT**

A Burndown Chart shows the actual and estimated amount of work to be done in a sprint. The horizontal x-axis in a Burndown Chart indicates time, while the vertical y-axis typically indicates story points.

**WHAT IS THE SPRINT REPORT?**

The Sprint Report includes the Burndown Chart, and lists the work completed, work not completed, and any work added after the sprint started.

**Git**

Git is Distributed [Version Control System](https://www.atlassian.com/git/tutorials/what-is-version-control) (DVCS). Unlike CVS or Subversion (SVN) repositories, [Git](https://www.atlassian.com/git) allows developers to create their own, personal copy of the team's repository, hosted alongside the main codebase. These copies are called forks and when work is complete on a fork, it's easy to bring changes back to the main codebase.

**Git branching**

There are other types of Git [branching](https://www.atlassian.com/agile/software-development/branching) besides task branching and they aren't mutually exclusive. You can create branches for a release, for example. This allows developers to stabilize and harden the work scheduled for a particular release, without holding up other developers who are working on future releases.

Once you've created a release branch, you'll need to regularly merge it into your master branch to ensure that your feature work makes it into future releases. To minimize overhead, it's best to create the release branch as close to the scheduled release date as possible.