Scrum Glossary

This glossary is meant to represent an overview of Scrum-related terms.

Some of the terms are not mandatory in Scrum, but have been added because they are commonly used in Scrum.

A second glossary lists Scrum related software development terminology.

Glossary of Scrum Terms

Burn-down Chart	A chart which shows the amount of work which is thought to remain in a backlog. Time is shown on the horizontal axis and work remaining on the vertical axis. As time progresses and items are drawn from the backlog and completed, a plot line showing work remaining may be expected to fall. The amount of work may be assessed in any of several ways such as user story points or task hours. Work remaining in Sprint Backlogs and Product Backlogs may be communicated by means of a burn-down chart. See also: Burnup Chart
Burn-up Chart	A chart which shows the amount of work which has been completed. Time is shown on the horizontal axis and work completed on the vertical axis. As time progresses and items are drawn from the backlog and completed, a plot line showing the work done may be expected to rise. The amount of work may be assessed in any of several ways such as user story points or task hours. The amount of work considered to be in-scope may also be plotted as a line; the burn-up can be expected to approach this line as work is completed.
Coherent/Coherence	The quality of the relationship between certain Product Backlog items which may make them worthy of consideration as a whole. See also: Sprint Goal.
Daily Scrum	Scrum Event that is a 15-minute time-boxed event held each day for the Developers. The Daily Scrum is held every day of the Sprint. At it, the Developers plans work for the next 24 hours. This optimizes team collaboration and performance by inspecting the work since the last Daily Scrum and forecasting upcoming Sprint work. The Daily Scrum is held at the same time and place each day to reduce complexity.
	A formal description of the state of the Increment when it meets the quality measures required for the product. The moment a Product Backlog item meets the Definition of Done, an Increment is born. The Definition of Done creates transparency by providing everyone a shared understanding of what work was completed as part of the Increment. If a Product Backlog item does not meet the Definition of Done, it cannot be released or even presented at the Sprint Review.

Developer	Any member of a Scrum Team, that is committed to creating any aspect of a usable Increment each Sprint regardless of technical, functional or other specialty.
Emergence	The process of the coming into existence or prominence of new facts or new knowledge of a fact, or knowledge of a fact becoming visible unexpectedly.
Empiricism	the philosophy that all knowledge originates in experience and observations. It's a cornerstone of the scientific method and underlies much of modern science and medicine. In the context of Scrum, empiricism refers to the idea that solving complex problems, or doing complex work, can only be done using an exploratory process rather than relying on predetermined plans.
Engineering standards	a shared set of development and technology standards that Developers apply to create releasable Increments of software.
Forecast (of functionality)	the selection of items from the Product Backlog Developers deems feasible for implementation in a Sprint.
Increment	Scrum Artifact that defines the complete and valuable work produced by the Developers during a Sprint. The sum of all Increments form a product.
Product Backlog	A Scrum Artifact that consists of an ordered list of the work to be done in order to create, maintain and sustain a product. Managed by the Product Owner.
Product Backlog refinement	the activity in a Sprint through which the Product Owner and the Developers add granularity to the Product Backlog.
Product Owner	Accountability in Scrum focussed on maximizing the value of a product, primarily by incrementally managing and expressing business and functional expectations for a product to the Developers.
Product Goal	The Product Goal describes a future state of the product which can serve as a target for the Scrum Team to plan against. The Product Goal is in the Product Backlog. The rest of the Product Backlog emerges to define "what" will fulfill the Product Goal.
Scrum	Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems as defined in the Scrum Guide
Scrum Board	a physical board to visualize information for and by the Scrum Team, often used to manage Sprint Backlog. Scrum boards are an optional implementation within Scrum to make information visible.
Scrum Guide	the definition of Scrum, written and provided by Ken Schwaber and Jeff Sutherland, co-creators of Scrum. This definition consists of Scrum's accountabilities, events, artifacts, and the rules that bind them together.
Scrum Master	Accountability within a Scrum Team focussed on guiding, coaching, teaching and assisting a Scrum Team and its environments in a proper understanding and use of Scrum.
Scrum Team	a self-managing team consisting of one Scrum Master, one Product Owner, and Developers.
Scrum Values	a set of fundamental values and qualities underpinning the Scrum framework; commitment, focus, openness, respect and courage.

Self-Managing	Scrum Teams are cross-functional, meaning the members have all the skills necessary to create value each Sprint. They are also self-managing, meaning they internally decide who does what, when, and how.
Sprint	Scrum Event that is time-boxed to one month or less, that serves as a container for the other Scrum events and activities. Sprints are done consecutively, without intermediate gaps.
Sprint Backlog	Scrum Artifact that provides an overview of the development work to realize a Sprint's goal, typically a forecast of functionality and the work needed to deliver that functionality. Managed by the Developers.
Sprint Goal	a short expression of the purpose of a Sprint, often a business problem that is addressed. Functionality might be adjusted during the Sprint in order to achieve the Sprint Goal.
Sprint Planning	Scrum Event that is time-boxed to 8 hours, or less, to start a Sprint. It serves for the Scrum Team to inspect the work from the Product Backlog that's most valuable to be done next and design that work into Sprint backlog.
Sprint Retrospective	Scrum Event that is set to a time-box of 3 hours, or less, to end a Sprint. It serves for the Scrum Team to inspect the past Sprint and plan for improvements to be enacted during future Sprints.
Sprint Review	Scrum Event that is set to a time-boxed of 4 hours, or less, to conclude the development work of a Sprint. It serves for the Scrum Team and the stakeholders to inspect the Increment of product resulting from the Sprint, assess the impact of the work performed on overall progress toward the Product Goal and update the Product backlog in order to maximize the value of the next period.
Stakeholder	a person external to the Scrum Team with a specific interest in and knowledge of a product that is required for incremental discovery. Represented by the Product Owner and actively engaged with the Scrum Team at Sprint Review.
Technical Debt	the typically unpredictable overhead of maintaining the product, often caused by less than ideal design decisions, contributing to the total cost of ownership. May exist unintentionally in the Increment or introduced purposefully to realize value earlier.
Values	When the values of commitment, courage, focus, openness and respect are embodied and lived by the Scrum Team, the *Scrum pillars* of transparency, inspection, and adaptation *come to life* and *build trust* for everyone. The Scrum Team members learn and explore those values as they work with the Scrum events, roles and artifacts.
Velocity	an optional, but often used, indication of the amount of Product Backlog turned into an Increment of product during a Sprint by a Scrum Team, tracked by the Developers for use within the Scrum Team.

Glossary of Scrum Software Development Terms

A/B Testing	extends the idea of hypothesis driven development by evaluating two or more different implementations to find out which one works best. Usually this is done by having different implementations and then route a part of our users to each of them. This allows to measure which implementation better supports the expected user behavior. A/B Testing is often combined with Feature Flags and Application Telemetry.
Acceptance Test-Driven Development (ATDD)	test-first software development practice in which acceptance criteria for new functionality are created as automated tests. The failing tests are constructed to pass as development proceeds and acceptance criteria are met.

Application Lifecycle Management(ALM)	holistic view on the management of software applications and systems, accounting for all stages of the existence of a software product.
Application Telemetry	Understanding how a product is used is a key factor for taking better decisions on where to invest. Application Telemetry can provide some insights to increase this understanding by showing usage statistics, performance parameters, user workflows and other relevant information.
Behavior-Driven Development (BDD)	agile software development practice adding to TDD the description of the desired functional behavior of the new functionality.
Blameless Postmortem	is to understand systemic factors that lead to an outage and identify learnings and actions that can help to prevent this kind of failure from recurring. This practice is based on the idea that in hindsight we usually know how the outage could have been prevented. But the past cannot be changed and therefore it is useless to discuss who should have done what, aka as blaming. But it is about shaping the future by learning from what just happened. What can we learn and how can we improve our process to make it more resilient?
Blue-Green Deployment	is a practice that helps reducing down-times while upgrading the system to a new version. It has other positive effects like fast rollbacks in case of emergency. It uses two identical environments. One environment (called blue to differentiate it from the other identical one) is handling all requests and executing all production operations. The other environment (green) can handle software updates and configuration changes without impacting production. Even tests can be executed on the green environment without risk. Once the green environment is ready, all requests are switched over to this one and it becomes the new blue environment. The previous blue environment at the same time becomes the green one and can be used for the next update.
Branching	creating a logical or physical copy of code within a version control system so that this copy might be changed in isolation.
Clean Code	software code that is expressed well, formatted correctly, and organized for later coders to understand. Clarity is preferred over cleverness.
Code Coverage	a measurement indicating the amount of product code that is exercised by tests.
Cohesion and Coupling	coupling refers to the interdependencies between modules, while cohesion describes how related the functions within a single module are.
Collective Code Ownership	a software development principle popularized by Extreme Programming holding that all contributors to a given codebase are jointly responsible for the code in its entirety.
Continuous Delivery	a software delivery practice similar to Continuous Deployment except a human action is required to promote changes into a subsequent environment along the pipeline.
Continuous Deployment	a software delivery practice in which the release process is fully automated in order to have changes promoted to the production environment with no human intervention.
Continuous Integration (CI)	agile software development practice popularized by Extreme Programming in which newly checked-in code is built, integrated and tested frequently, generally multiple times a day.

Continuous Testing	Traditional approaches of quality assurance are often based on getting all implementation finished before verifying the latest build of the product before delivering it to production. In contrast continuous testing is a practice of integrating testing as a fundamental and ongoing part of development. It helps to identify and fix issues much earlier and so lowers risk drastically. Continuous testing is especially powerful if combined with practices like test automation, clean code and others which help to reduce regression testing efforts.
Cycle Time	is the time between working on an item that has been started and the item is finished (usually delivered to real end-users). Cycle Time defines how fast work can flow through a system and minimizing Cycle Time helps not only to make the system more efficient but also to increase predictability and the ability to quickly respond to changes or new insights.
Cyclomatic Complexity	a measure of code complexity based on the number of independent logical branches through a codebase. Cyclomatic complexity is expressed as a simple integer.
Cross-functional	characteristic of a team holding that all the skills required to successfully produce a releasable Increment in a Sprint are available within the team, where releasable refers to making the software available in production.
DevOps	an organizational concept serving to bridge the gap between development and operations, in terms of skills, mind-set, practices and silo-mentality. The underlying idea is that developers are aware of, and in daily work consider implications on operations, and vice versa.
Dev & Ops collaboration	is at the core of the DevOps movement. Instead of separating development and operations, collaboration is key. Instead of developing something and then make running this in production someone else's problem, we try to achieve end-to-end responsibility. This not only helps with smoothening the delivery process, but it also closes the feedback loop. A closer collaboration strengthens learning how we could support robust operation already in the design and implementation operations.
DRY (don't repeat yourself)	software development principle to avoid repetition of the same information in one system, preventing the same code from being produced multiple times on a codebase.
Engineering Standards	a shared set of development and technology standards that the Developers apply to create releasable Increments of software, and against which those Developers can inspect and adapt.
Error Culture	How mistakes are handled has an important impact on an organizations ability to innovate. If people feel that errors are something negative and try to avoid them, they might be much less likely to take a risk and try something new. Instead encouragement for experimentation and learning is important while at the same time considering how to tame risks and decrease the impact of failures.
Extreme Programming (XP)	agile software development framework with an extreme focus on programming and taking engineering practices to an extreme in order to create and release high quality code. Highly complementary to the Scrum framework.
Feature Flags/Feature Toggle	software development practice that allows dynamically turning (parts of) functionality on and off without impacting the overall accessibility of the system by its users.
Hypothesis Driven Development	The basic idea behind Hypothesis Driven Development is that in a complex environment we do not know where to invest in order to achieve the highest possible value: we formulate hypotheses about that. Once we accept uncertainty and agree that the assumptions our plans are based on can be wrong, it makes sense to change our approach to the development of new features. Validating our assumptions gets high priority and finding small and fast experiments to get more insights becomes an important part of work.

Infrastructure as Code	Instead of setting up and configuring infrastructure and environments, this process can be automated by scripts and parameter files. While this approach is not faster for the initial setup of the infrastructure (it might even be slower), it provides a lot of advantages. The scripts and configuration files can be stored in version control together with the source code of the software. This allows to create exactly the matching environment for a given version of the software. Changes to the environment are documented in version control as they are no longer executed on the environment directly but by changing and executing a script. And new instances of an exact copy of the production environment can easily be created, for example for testing purpose.
Mean Time to Detect (MTTD)	is the time it takes to identify that there is a problem. The MTTD should not be determined by the first call of an angry customer at the hotline. Monitoring tools can help to reduce it.
Mean Time to Recover (MTTR)	is the average time it takes from when a problem occurs until the problem is fixed and the system is back to normal operations. There are various techniques helping with MTTR as defensive programming and self-healing systems that can switch to an emergency mode to keep the basic functionality of the system up and running. As MTTD is usually a significant portion of MTTR, reducing MTTD will also help with MTTR.
Minimum Viable Product	One practice to achieve Hypothesis Driven Development is the Minimum Viable Product (MVP). The MVP is the smallest implementation of our product or a feature which allows to learn more about how users will react to it or technical behavior like performance
Monitoring	Obviously, it is helpful to know the current state of your system. While most systems support logging to analyze what happened during an outage or a problem, monitoring is used to check the current state continuously. Once one or more parameters get out of a healthy range, alarms can be triggered to initiate some actions
Pair Programming	agile software development practice popularized by Extreme Programming in which two team members jointly create new functionality.
Refactoring	agile software development practice popularized by Extreme Programming in which code is adjusted within the codebase without impacting the external, functional behavior of that code.
Release-Pipelines	Automating the steps from code commit into version control to delivery in production help increasing speed and reliability of this process. This practice is often referred as Release-Pipelines as this pictures the ideal of a steady stream of changes delivered.
Scout Rule	the practice of always leaving the codebase in a little better state than it was found before modifications. A means to progress towards Clean Code.
Specification by Example	agile software development practice based on TDD and ATDD that calls for using realistic examples from past experience instead of untested or abstract statements in the description of the desired functional behavior.
Test-Automation	Automating tests is not only a way to increase the quality of a product, it also helps to reduce cycle time. If you see test execution as one form of feedback, automated tests help to get this feedback much earlier. This lowers the effort of fixing issues and allows to deliver releasable increments much faster.
Test-Driven Development (TDD)	test-first software development practice in which test cases are defined and created first, and subsequently executable code is created to make those tests pass. The failing tests are constructed to pass as development proceeds and tests succeed.

Testing in Production	What sounds like a very dangerous approach can help not only to reduce cycle time but also make your tests more reliable. It means to execute various tests right in the production environment. To reduce the risk of affecting production operations with untested and failing functionality, various technologies can be used to make the new functionality only available to the test process or a very small set of users while other users will use the previous functionality. The change will be rolled out to all users once the test was successful. This practice will help you to get rid of various test stages like QA, UAT, Staging etc. And as the tests run in the real production environment, you can eliminate the risk of using a test environment that behaves slightly different from your production environment.
User Story	agile software development practice from Extreme Programming to express requirements from an end user perspective, emphasizing verbal communication. In Scrum, it is often used to express functional items on the Product Backlog.
Unit Test	low-level technical test focusing on small parts of a software system that can be executed fast and in isolation. The definition and boundaries of a 'unit' generally depends on the context and is to be agreed upon by the Developers
Vertical Teams	In traditional organizations, there are many different teams and departments involved in the process from a customer providing feedback or suggesting an improvement to delivering the new version to customers. Sales & Marketing, Product Management, Development, Quality Assurance, Operations and more jump to mind. But each different department or team means there is a handoff. Handoffs tend to be slow and error-prone. In contrast a vertical team combines all the necessary competencies to handle the whole process end-to-end. In such scenarios teams typically are only responsible for small parts of the whole product. So instead of splitting the organization into horizontal layers (departments) this approach suggests slicing the product.