### Scrum terms

|  |  |
| --- | --- |
| **Glossary** |  |
| **English Name** | **Description** |
| **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 apart 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.  Acceptance Test Driven Development (ATDD) involves team members with different perspectives (customer, development, testing) collaborating to write acceptance tests in advance of implementing the corresponding functionality. |
| **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. |
| **Age** | The amount of time between when a work item started and the current time. This applies only to items that are still in progress. |
| **Acceptance Testing** | An acceptance test is a formal description of the behavior of a software product, generally expressed as an example or a usage scenario. A number of different notations and approaches have been proposed for such examples or scenarios. |
| **AntiPattern** | Antipatterns are common solutions to common problems where the solution is ineffective and may result in undesired consequences. |
| **Automated Build** | In the context of software development, build refers to the process that converts files and other assets under the developers' responsibility into a software product in its final or consumable form. The build is automated when these steps are repeatable, require no direct human intervention, and can be performed at any time with no information other than what is stored in the source code control repository. |
| **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 knowhow 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 aversion control system so that this copy might be changed in isolation. |
| **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. |
| **Backlog Refinement** | Backlog grooming is when the product owner and some, or all, of the rest of the team refine the backlog on a regular basis to ensure the backlog contains the appropriate items, that they are prioritized, and that the items at the top of the backlog are ready for delivery. |
| **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. |
| **Collective Ownership** | Collective code ownership is the explicit convention that every team member can make changes to any code file as necessary: either to complete a development task, to repair a defect, or to improve the code's overall structure. |
| **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 deployment aims to reduce the time elapsed between writing a line of code and making that code available to users in production. To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code. |
| **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 Integration is the practice of merging code changes into a shared repository several times a day in order to release a product version at any moment. This requires an integration procedure which is reproducible and automated. |
| **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.  The amount of elapsed time between when a workitem starts and when a work item finishes. |
| **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. |
| **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. |
| **Cumulative Flow Diagram (CFD)** | A chart to visualize theflow of work through a process. |
| **Cycle time scatterplot** | A chart to show the total elapsedtime it took for individual items to move from one point of theWorkflow to another - usually from start to finish. |
| **CRC Cards** | Class Responsibility Collaborator (CRC) Cards are an object oriented design technique teams can use to discuss what a class should know and do and what other classes it interacts with. |
| **Customer Development** | Customer development is a four-step framework, originally identified by Steve Blank, to discover and validate that you have identified a need(s) that customers have built the right product to satisfy that customer’s need(s), tested the correct methods for acquiring and converting customers, and deployed the right resources in the organization to meet the demand for the product. The framework provides a way to use a scientific approach to validate assumptions about your product and business. |
| **Capabilities** | Course grained requirements, Large Epics, or Features. |
| **Definition of Done (DoD)** | a shared understanding of the expectations that software must live up to in order to be releasable into production, with a purpose of providing transparency over the software created.  is 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.  The definition of done is an agreed upon list of the activities deemed necessary to get a product increment, usually represented by a user story, to a done state by the end of a sprint. |
| **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. |
| **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. |
| **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 plan 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. |
| **Daily Meeting** | The daily meeting is one of the most commonly practiced Agile techniques and presents opportunity for a team to get together on a regular basis to coordinate their activities. |
| **Definition of Ready** | Definition of Ready involves creating clear criteria that a user story must meet before being accepted into an upcoming iteration. This is typically based on the INVEST matrix. |
| **Demo (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. |
| **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. a shared set of development and technology standards that Developers apply to create releasable Increments of software. |
| **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. Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development. |
| **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. |
| **Epic** | An epic is a large user story that cannot be delivered as defined within a single iteration or is large enough that it can be split into smaller user stories.  An epic is a large user story that cannot be delivered as defined within a single iteration or is large enough that it can be split into smaller user stories.  Large User Stories that have reasonably defined requirements |
| **Estimation** | In software development, an "estimate" is the evaluation of the effort necessary to carry out a given development task; this is most often expressed in terms of duration.  In software development, an "estimate" is the evaluation of the effort necessary to carry out a given development task; this is most often expressed in terms of duration. |
| **Exploratory Testing** | Exploratory testing is, more than strictly speaking a "practice," a style or approach to testing software which is often contrasted to "scripted testing." |
| **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. |
| **Forecast (of functionality)** | the selection of items from the Product Backlog Developers deems feasible for implementation in a Sprint. |
| **Facilitation** | A facilitator is a person who chooses or is given the explicit role of conducting a meeting. |
| **Frequent Releases** | An Agile team frequently releases its product into the hands of end users, listening to feedback, whether critical or appreciative. |
| **Given – When – Then** | The Given-When-Then formula is a template intended to guide the writing of acceptance tests for a User Story: (Given) some context, (When) some action is carried out, (Then) a particular set of observable consequences should obtain. |
| **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. |
| **Heartbeat Retrospective** | The team meets regularly to reflect on the most significant events that occurred since the previous such meeting, and identify opportunities for improvement. |
| **Increment** | a fully functional piece of working software, living up to the Definition of Done, that adds to previously created Increments, where the sum of all Increments - as a whole - form a product. The moment a Product Backlog item meets the Definition of Done, an Increment is born. Scrum Artifact that defines the complete and valuable work produced by the Developers during a Sprint. The sum of all Increments form a product. |
| **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. |
| **Information Radiators** | Information radiator is the term for any of a number of visual displays which a team places in a highly visible location, so that all team members can see the latest information at a glance. |
| **INVEST** | The acronym INVEST stands for a set of criteria used to assess the quality of a user story. If the story fails to meet one of these criteria, the team may want to reword it. |
| **Iteration** | An iteration is a timebox during which development takes place. The duration may vary from project to project and is usually fixed. |
| **Iterative Development** | Agile projects are iterative insofar as they intentionally allow for "repeating" software development activities, and for potentially "revisiting" the same work products (the phrase "planned rework" is sometimes used; refactoring is a good example). |
| **Incremental Development** | In an Agile context, Incremental Development is when each successive version of a product is usable, and each builds upon the previous version by adding user-visible functionality. |
| **Integration** | Integration (or "integrating") refers to any efforts still required for a project team to deliver a product suitable for release as a functional whole. |
| **Kanban** | The Kanban Method is a means to design, manage, and improve flow systems for knowledge work. The method also allows organizations to start with their existing workflow and drive evolutionary change. They can do this by visualizing their flow of work, limit work in progress (WIP) and stop starting and start finishing. A strategy for optimizing the flow of value through a process that uses a visual, work in-progress limited pullsystem. |
| **Kanban Board** | A Kanban Board is a visual workflow tool consisting of multiple columns. Each column represents a different stage in the workflow process. |
| **Lead Time** | Lead Time is the time between a customer order and delivery. In software development, it can also be the time between a requirement made and its fulfillment. |
| **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 Simulations using randomsampling to produce hundreds of thousands of possibleoutcomes which are then analyzed to get probabilities of thedifferent outcomes occurring. |
| **Milestone Retrospective** | A Milestone Retrospective is a team's detailed analysis of the project's significant events after a set period of time or at the project's end. |
| **Minimum Marketable Feature (MMF)** | A Minimum Marketable Feature is a small, self-contained feature that can be developed quickly and that delivers significant value to the user. |
| **Mob Programming** | Mob Programming is a software development approach where the whole team works on the same thing, at the same time, in the same space, and at the same computer. |
| **Mock Objects** | Mock Objects (commonly used in the context of crafting automated unit tests) consist of instantiating a test-specific version of a software component. |
| **Niko-niko Calendar** | A Niko-niko Calendar is updated daily with each team member's mood for that day. Over time the calendar reveals patterns of change in the moods of the team, or of individual members. |
| **Open Space** | When you bring together a group of folks, empower them to co-create their own learning, provide just enough structure to ensure progress, and hit the Start button – that’s Open […] |
| **Product Backlog** | A product backlog is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome.  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** | The product owner is a role on a product development team responsible for managing the product backlog in order to achieve the desired outcome that a product development team seeks to accomplish.  Role in Scrum accountable for 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. |
| **Policies** | Explicit rules about how work flows through each state (which may include items from a Scrum Team's Definition of Done and pull policies between stages) |
| **Probabilistic forecasting** | One that acknowledges a wide array of possible outcomes and assigns a probability, or likelihood of happening, to each. This makes it the best method for forecasting in uncertain situations. |
| **Pair Programming** | Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "driver", the other, also actively involved in the programming task but focusing more on overall direction is the "navigator"; it is expected that the programmers swap roles every few minutes or so.  agile software development practice popularized by Extreme Programming in which two team members jointly create new functionality. |
| **Personas** | Personas are synthetic biographies of fictitious users of the future product. |
| **Planning Poker** | An approach to estimation used by Agile teams. Each team member "plays" a card bearing a numerical value corresponding to a point estimation for a user story. |
| **Points (estimates in)** | Agile teams generally prefer to express estimates in units other than the time-honored "man-hours." Possibly the most widespread unit is "story points." |
| **Project Chartering** | A high-level summary of the project's key success factors displayed on one wall of the team room as a flipchart-sized sheet of paper. |
| **Potential Releasable Increment (PRI)** | “Potentially Releasable” signifies that the product increment can be launched to users at the end of a Sprint, meeting all the criteria for release. |
| **Quick Design Session** | When "simple design" choices have far-reaching consequences, two or more developers meet for a quick design session at a whiteboard. |
| **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. Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behavior. |
| **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. |
| **Ready** | a shared understanding by the Product Owner and the Developers regarding the preferred level of description of Product Backlog items introduced at Sprint Planning. |
| **Refinement** | See Product Backlog Refinement |
| **Relative Estimation** | Relative estimation is one of the several distinct flavors of estimation used in Agile teams, and consists of estimating tasks or user stories, not separately and in absolute units of time, […] |
| **Rules of Simplicity** | Rules of Simplicity is a set of criteria, in priority order, proposed by Kent Beck to judge whether some source code is "simple enough." |
| **Ready User Stories** | Small, clear, and testable stories that can be implemented in a single Sprint, Detailed requirements and acceptance criteria |
| **Retrospectiv/lessons learned** | A sprint retrospective brings your scrum team together to discuss the previous sprint. The purpose is not to evaluate work outcomes but to talk about the interactions, tools, and processes the team used during the latest period of work. Retrospectives are usually held at the end of each sprint. |
| **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. |
| **Scrum** | a framework to support teams in complex product development. Scrum consists of Scrum Teams and their associated accountabilities, events, artifacts, and rules, as defined in the Scrum Guide™.  Scrum is a process framework used to manage product development and other knowledge work. Scrum is empirical in that it provides a means for teams to establish a hypothesis of how they think something works, try it out, reflect on the experience, and make the appropriate adjustments. That is, when the framework is used properly.  Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems as defined in the Scrum GuideTM. |
| **Scrum Board** | a board to visualize information within the Scrum Team primarily, often used to manage Sprint Backlog. Scrum boards are a complimentary practice within Scrum to make information visible and thereby increase transparency.  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. The Scrum Guide consists of Scrum’s accountabilities, events, artifacts, and the rules that bind them together. |
| **Scrum Team (ST)** | A self-managing team consisting of a Product Owner, Development Team and Scrum Master. |
| **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. |
| **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. |
| **Scrum Master (SM)** | Role within a Scrum Team accountable for guiding, coaching, teaching and assisting a Scrum Team and its environments in a proper understanding and use of Scrum.  The scrum master is the team role responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use.  The scrum master is the team role responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use. |
| **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. |
| **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. A sprint is an iteration. |
| **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.  A sprint backlog is the subset of product backlog that a team targets to deliver during a sprint to accomplish the sprint goal and progress toward an outcome. |
| **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. |
| **Story Points** | Story points in Scrum are units of measurement used to estimate the effort required to complete a story. When planning for an upcoming sprint, Scrum teams use story point estimation to gauge how much effort is needed to develop a new software feature or update. |
| **Service Level Expectation (SLE)** | A forecast of how long it should take a given item to flow from start to finish within the Scrum Team’s Workflow. |
| **Scrumban** | Scrumban is a mixture of the Scrum Method and the Kanban Method. |
| **Sign Up For Tasks** | Members of an Agile development team normally choose which tasks to work on, rather than being assigned work by a manager. |
| **Story Mapping** | Story mapping consists of ordering user stories along two independent dimensions based on the order activities occur and sophistication of implementation. |
| **Story Splitting** | Splitting consists of breaking up one user story into smaller ones, while preserving the property that each user story separately has measurable business value. |
| **Sustainable Pace** | The team aims for a work pace that they would be able to sustain indefinitely. |
| **Size** | an estimate of the work to complete a user story in consistent units across the backlog |
| **Scrum of Scrums** | A technique to scale Scrum up to large groups (over a dozen people), consisting of dividing the groups into Agile teams of 5-10. |
| **Simple Design** | A team adopting the "simple design" practice bases its software design strategy on a set of "simple design" principles. |
| **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.  Test-driven development (TDD) is a style of programming where coding, testing, and design are tightly interwoven. Benefits include reduction in defect rates. |
| **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. |
| **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. |
| **Throughput** | The number of work items finished per unit of time. |
| **Task Board** | A basic task board is divided into three columns labeled "To Do," "In Progress," and "Done." Cards are placed in the columns reflecting the current status. |
| **Team** | A "team" in the Agile sense is a small group of people, assigned to the same project or effort, nearly all of them on a full-time basis. |
| **Team Room** | The team has the use of a dedicated space for the duration of the project, set apart from other groups' activities. |
| **The Three C’s** | Card, Conversation, Confirmation is a formula that captures the social nature of user stories. |
| **Three Amigos** | Three amigos refers to the primary perspectives to examine an increment of work before, during, and after development. Those perspectives are: Business - What problem are we trying to solve? Development - How might we build a solution to solve that problem? Testing - What about this, what could possibly happen? |
| **Three Questions** | The daily meeting is structured around the following three questions: What have you completed? What will you do next? What is getting in your way? |
| **Timebox** | A timebox is a previously agreed period of time during which a person or a team works steadily towards completion of some goal. |
| **T-shaped Development team** | T shaped Development teams or Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team.” The “T-shaped” metaphor comes from the idea that an individual can possess deep skills in a few areas as well as a broader range of shallower skills. |
| **T-shaped competence** | In a Scrum team, a T-shaped member is someone with a strong primary skill or area of expertise, often referred to as their “vertical” skill. This skill could be in programming, design, quality assurance, or any other specialization related to software development |
| **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. |
| **Ubiquitous Language** | Teams use a ubiquitous language to use the vocabulary of a business in the requirements, design discussions and source code for a software product. |
| **Usability Testing** | Usability testing is an empirical, exploratory technique to answer questions such as "how would an end user respond to our software under realistic conditions?" |
| **User Stories** | In consultation with the customer or product owner, the team divides up the work to be done into functional increments called "user stories."  a description of an element of the solution that is to be built: The product’s unit feature or functionality |
| **User Story Template** | The "role-feature-reason" template is one of the most commonly recommended aids to write user stories: As a ... I want ... So that ... |
| **Unit Testing** | A unit test is a short program fragment which exercises some narrow part of the product's source code and checks the results. |
| **Velocity** | an optional but often used indication of the amount of Product Backlog turned into an Increment of product during a Sprint. It is tracked by the Developers for use within the Scrum Team. Velocity is the total effort estimates associated with user stories that were completed during an iteration. the “size” of work that the team(s) can accomplish in a single sprint. |
| **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. |
| **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. Download the Scrum Values Poster. |
| **Version Control** | Version control is not merely "good practice" but an enabler of a number of Agile practices, such as continuous integration. |
| **WIP** | The number of work items started but not finished. |
| **WIP Aging chart** | A chart to visualize the age of active workitems on the Scrum Team Workflow. |
| **Work In Progress (WIP) Limit** | Policies to limit work in progress on the Workflow. The following terms are used in the - Metrics for Predictability or from Actionable Agile documentation. |
| **Work item** | When using Kanban within Scrum, a work item is a synonym of a product backlog item. |
| **Workflow** | The explicit shared understanding of flow among the Scrum Team. |