Agile Glossary of Terms

Acceptance Criteria: Those criteria by which a work item (usually a user story) is judged successful or not; usually "all or nothing", it is "done", or "not done". In the traditional programming paradigm, this refers to criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity. Acceptance criteria are developed to identify when the user story has been implemented, and the user story will be able to achieve their goals.

Acceptance Testing: Formal testing conducted to determine whether or not a user story satisfies its acceptance criteria and to enable the customer to determine whether or not to accept it.

Agile: An umbrella term for a variety of best practices in software development. Agile software development supports the practice of shorter software delivery. Specifically, Agile calls for the delivery of software in small, short increments rather than in the typically long, sequential phases of a traditional waterfall approach. More a philosophy than a methodology, Agile emphasizes this early and continuous software delivery, as well as using collaborative teams, and measuring progress with working software. There are many specific methodologies that fall under this category. Some examples are: Scrum, eXtreme Programming (XP), and Kanban.

Agile Manifesto: A statement of the principles and values that support the ideals of Agile Software Development; the manifesto was drafted in February 2001 at the Snowbird Ski Resort located in the state of Utah. Users of Agile can become signatories of this manifesto at http://www.agilemanifesto.org

The Agile Manifesto: We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Architecture: In general, architecture is a set of structures for a product. These structures are comprised of elements, relations among them, and properties of both. In a service context, the architecture is often applied to the service or system. Note that functionality is only one aspect of the product. Quality attributes, such as responsiveness, reliability, and security, are also important to consider. Structures provide the means for highlighting different portions of the architecture. Architectural patterns an organization, program, or team might consider to support the rapid delivery of Agile software development include, among other things, micro-services and evolutionary design. There are many types of architecture. For example, a functional architecture is a hierarchical arrangement of functions, their internal and external functional interfaces and external physical interfaces while an information security architecture is an embedded, integral part of the enterprise that describes the structure and behavior of the enterprise's security processes, information security systems, personnel, and organizational subunits, showing their alignment with the enterprise's mission and strategic plans.

Backlog: The backlog is a list of user stories to be addressed by working software. If new requirements or defects are discovered, these can be stored in the backlog to be addressed in future iterations. A complete list of work items ordered from the highest priority to the lowest priority. Backlogs were developed in the context of the Scrum methodology but are now widely used in many Agile frameworks. Backlogs include both functional and non-functional work (often reflected as user stories), including technical team-generated stories. Backlogs can occur at varying levels. For example, a product backlog is a high-level backlog that contains all the requirements for the entire development project and an iteration backlog includes a list of user stories intended for that iteration.

Backlog Grooming: Similar to a requirements scrub in the traditional programming paradigm, backlog grooming refers to the process of adding detail and revisiting the order and estimates assigned to work in the backlog. This is also called "backlog refinement".

Behavior Driven Development: Behavior driven development (or BDD) is an agile software development technique that encourages collaboration between developers, QA and non- technical or business participants in a software project. BDD focuses on obtaining a clear understanding of desired software behavior through discussion with stakeholders. It extends TDD by writing test cases in a natural language that non-programmers can read.

Bottleneck: A bottleneck is a sort of congestion in a system that occurs when workload arrives at a given point more quickly than that point can handle it.

It is metaphorically derived from the flowing of water through a narrow-mouthed bottle where the flow of water is constrained by the size of its neck.

Bugs: A software bug is a problem causing a program to crash or produce invalid output. It is caused by insufficient or erroneous logic and can be an error, mistake, defect or fault.

Burn-Down Chart: A visual tool displaying progress via a simple line chart representing the remaining work (vertical axis) over time (horizontal axis). It shows where the team stands regarding completing the tasks that comprise the backlog items that the goals of the iteration. Related to the burn-up chart, except burn-down charts display remaining work instead of work accomplished.

Burn-Up Chart: A visual tool displaying progress via a simple line chart representing work accomplished (vertical axis) over time (horizontal axis). Burn-up charts are also typically used at a release level and iteration levels. They are related to the burn-down chart except they display accomplished work instead of remaining work.

Business sponsor: Owns the business case for the project and is responsible for business solution. They are usually the most senior person on the project and typically allow the project to progress without interference; generally, only getting involved with escalated issues.

Cadence: (see: velocity)

Capacity: The quantity of resources available to perform useful work.

Champion: Spreads Agile principles and continually makes adjustments to Agile practices that suit the environment for successful outcomes. Their goal is to assist with the Agile adoption and transformation and influence others, relevant to the Agile process.

Chicken: Someone on an Agile project who is involved but not committed (see also Pigs). Chickens should not be part of the core project team but may have input or feedback. Chickens are often interested stakeholders, managers, and executives. As these individuals are not directly involved or accountable, it is encouraged the Chickens participation in the process is limited to observation only.

Coding Standards: Coding standards are used even outside of Agile, maybe rewrite to: An agreed upon set of standards for programming style, practices and methods. Coding standards keep the code consistent and easy for the entire team to read and refactor. The concept is that code that looks the same encourages collective ownership.

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 code ownership, as the name suggests, is the explicit convention that "every" team member is not only allowed, but in fact has a positive duty, to make changes to "any" code file as necessary: either to complete a development task, to repair a defect, or even to improve the code's overall structure.

Complexity Point: Units of measure used to estimate development work in terms of complexity but not effort. For Agile projects, effort is typically measured in story points.

Continuous Delivery: Continuous delivery is one of the principles of the Agile Manifesto. Continuous delivery builds on continuous integration by taking the step of orchestrating multiple builds, coordinating different levels of automated testing, and in advanced cases moving the code into a production environment.

Continuous Deployment: Continuous deployment builds upon continuous delivery and is 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: Teams practicing continuous integration seek two objectives: (1) minimize the duration and effort required by "each" integration episode; (2) be able to deliver "at any moment" a product version suitable for release. In practice, this dual objective requires an integration procedure which is "reproducible" at the very least, and in fact largely "automated". This is achieved through version control tools, team policies and conventions, and tools specifically designed to help achieve continuous integration.

Could Have: Those features that are not critical for the project. While these features have a higher priority than Nice to Have features, they do not need to be delivered as part of the requirements. (See also: Should Have, Must Have, and Nice to Have.)

Customer: Synonymous with business sponsor because they are ultimately the buyers of the solution. They want continuous improvement of products and services. The ultimate buyer of the solution. They

are an integral part of the development and have specific responsibilities in different Agile frameworks, relevant to the Agile product.

Cycle Time: The amount of time needed to complete a feature or user story during the development process.

Daily Backlog: The backlog requirements for the iteration as they are allocated into daily work assignments.

Daily Stand Up Meeting: A brief, daily communication and planning forum where the development team and other stakeholders evaluate the health and progress of the iteration.

Definition of Done: A predefined set of criteria defined and displayed by the team that must be met before a work item is considered complete. It should prevent the accumulation of technical debt that organically arises when it is defined colloquially and loosely.

Demo (Demonstration): At the end of each iteration the development team performs a demonstration of the functionality that has been completed during the iteration. The demo is a ceremony for the business stakeholder to provide feedback on the product's development.

Epic: A large user story, perhaps a few too many months in size, that can span an entire release or multiple releases. Epics are useful as placeholders for large requirements. Epics are progressively refined into a set of smaller user stories at the appropriate time.

Estimation: The process of agreeing on a size measurement for the stories in a product backlog; on agile projects estimation is done by the team responsible for delivering the work, usually using a planning game (see Planning Poker).

Evolutionary Development: The "evolutionary" strategy develops a system in builds, but differs from the incremental strategy in acknowledging that the user need is not fully understood and all requirements cannot be defined up front. In this strategy, user needs and system requirements are partially defined up front, then are refined in each succeeding build.

eXtreme Programming (XP): A type of Agile software development based on the values of communication, simplicity, feedback, and respect. Some of XP's core principles are: planning poker, test-driven development, refactoring, pair programming, collective ownership, continuous integration, coding standards, and sustainable pace.

Feature: Functional or non-functional distinguishing characteristic of a system, it can be an enhancement to an existing system. Features are a customer-understandable, customer-valued piece of functionality that serves as a building block for prioritization, planning, estimation, and reporting.

Five Levels of Agile Planning: The five levels of Agile planning are Vision, Roadmap, Release, Iteration (or Sprint), and Daily. The top level (Vision) represents the "big picture" of the overall effort and thus the planning at this level encompasses more strategic product information and fewer details on the product specifics.
Working through to the bottom level, more details are included in the produced plans, so that in whole, the five levels of Agile planning represents a holistic understanding of what we are building, why we are undertaking the effort, and how we plan to deliver.

Framework: collection of values, principles, practices, and rules that form the foundation of development

Function Point: A unit of measure for functional size that looks at the logical view of the software code accounting for external inputs, external outputs, external inquiries, external interface files, and internal logical files.

Integration Testing: The phase in software testing in which individual software modules are combined and tested as a group. It typically occurs after unit testing and before validation testing.

Information Radiator: A group of artifacts that is used to communicate project status to the team and other stakeholders; information radiators are an important part of maintaining transparency and visibility into the team's progress.

Invest: INVEST is an acronym that defines a simple set of rules used in creating well-formed User Stories.

> Independent: Stories should not be dependent on other stories.
> Negotiable: Too much explicit detail regarding particulars and solutions. Stories should capture the essence of the requirement and should not represent a contract on how to solve it.
> Valuable: Stories should clearly illustrate value to the customer.
> Estimable: Stories should provide just enough information so they can be estimated. It is not important to know the exact way that a particular problem will be solved, it must be understood enough to provide a high-level estimate.
> Small: Stories should strive to be granular enough in scope that they may be completed in as little time as possible, from a few weeks to a few days.
> Testable: Stories need to be understood well enough so that a test can be defined for it. An effective way to ensure testability is to define user acceptance criteria for all user stories.

Iteration: A predefined, time-boxed and recurring period of time in which working software is created. Instead of relying on extensive planning and design, iterations rely on rework informed by user feedback and learning.

Kanban: The term "Kanban", pronounced /ˈkɑnˈbɑn/, is Japanese and derived from roots which literally translate as "visual board". The focus of Kanban is to optimize the throughput of work by visualizing its flow of work through the process, limiting work in progress (WIP), and explicitly identifying policies for the flow of work. Kanban has distinct differences from other popular agile methodologies, primarily the fact that it is not based on time-boxed iterations, but rather allows for continuous prioritization and delivery of work.

Kanban Board: Unlike a task board, the Kanban board is not "reset" at the beginning of each iteration; its columns represent the different processing states of a "unit of value", which is generally (but not necessarily) equated with a user story; and each column may have associated with it a "WIP limit" (for "work in process" or "work in progress"). The priority is to clear current work-in-process, and team members will "swarm" to help those working on the activity that's blocking flow.

Kanban Method: An approach to continuous improvement which relies on visualizing the current system of work scheduling, managing "flow" as the primary measure of performance, and whole-system optimization - as a process improvement approach, it does not prescribe any particular practices. Agile teams employing a Kanban method may deemphasize the use of iterations, effort estimates and velocity as a primary measure of progress; rely on measures of lead time or cycle time instead of velocity; and replace the task board with a "Kanban Board."

Lead Time: The amount of time needed to complete a feature or user story from customer request to delivery (includes wait time).

Lean: Lean software development is a translation of Lean manufacturing and Lean IT principles and practices to the software development domain. Adapted from the Toyota Production System and is a set of techniques and principles for delivering more values with the same or less resources by eliminating waste across organizations and business processes

Minimally Viable Product: The simplest version of a product that can still be released. A minimally viable product should have enough value that it is still usable, it demonstrates future benefit early on to retain user buy in, and provides a feedback loop to help guide future development.

Must Haves: Those features that are critical for the project; these are the features that must be delivered as part of the requirements. In addition to must have features, there are also Should Have, Could Have, and Nice to Have features. (See also: Should Have, Could Have, and Nice to Have.)

Nice to Have: Those features that are not critical for the project's success. These are the features that are developed if there is enough time or money to develop. (See also: Should Have, Could Have, and Must Have.)

Pair Programming: Two developers working side by side to develop code. This method of programming provides a real-time code review; allowing one developer to think ahead while the other thinks about the work at hand, and supports cross-training. The concept can also be extended to pair designing and pair unit testing to provide real time peer reviews. Pair programming is a fundamental part of XP.

Peer Inspections: A form of code review that occurs after the code is complete to ensure consistency.

Pig: Someone on an Agile project who is committed and in impacted by the outcome (see also Chicken). Pigs are encouraged to wholly participate in the process, as they are accountable for the expectations set by their involvement and commitments.

Planning Poker: A consensus-based technique used to estimate how long a certain amount of work will take to complete. This is related to a group estimation technique known as wide-band Delphi from traditional planning.

Product Backlog: All of the known features or stories that may be implemented throughout the project; these are typically high-level requirements with high level estimates provided by the product stakeholders. The requirements are listed on the backlog in priority order and maintained by the product owner.

Product Owner: The "voice of the customer", accountable for ensuring business value is delivered by creating customer-centric items (typically user stories), ordering them, and maintaining them in the backlog. In Scrum, the product owner is the sole person/entity responsible for managing the backlog. The product owner's duties typically include clearly expressing the backlog items, ordering the backlog items to reflect goals and missions, keeping the backlog visible to all, optimizing the value of development team work, ensuring that the developers fully understand the backlog items and deciding when feature is "done". A product owner should be available to the team in a within a reasonable time for both decision making and empowerment.

Progressive Elaboration: An iterative approach where planning occurs in cycles rather that all up front. Projects, which use progressive elaboration, typically do planning, and execution and then repeat the process.

Project: the result of a development effort

Quality Attribute: A factor that specifies the degree of an attribute that affects the quality that the system or software must possess, such as performance, modifiability, or usability.

Regression Testing: A type of software testing that verifies that software that was previously developed and tested still performs correctly after it was changed or interfaced with other software. These changes may include software enhancements, patches, configuration changes, etc. During regression testing, new software bugs or regressions may be discovered.

Refactoring: Refactoring was popularized as a practice in XP, but was used prior to XP and is now used by most Agile methods as a normal part of the development process. Refactoring involves modifying/revising code to improve performance, efficiency, readability, or simplicity without affecting functionality. Generally considered part of the normal development process, refactoring improves software longevity, adaptability, and maintainability over time.

Release: A planning segment of requirements (typically captured as user stories in the backlog), along with execution estimates. In the traditional programming paradigm, a release is a delivered version of an application which may include all or part of an application.

Requirements: A condition or capability needed by a user to solve a problem or achieve an objective.

Requirements Scrub: (see Backlog Grooming)

Retrospective: A team meeting that occurs at the end of every iteration to review lessons learned and to discuss how the team can improve in the future. The retrospective is an integral part of Agile planning and process/product improvement; typically, retrospectives occur at the end of every iteration. During each retrospective, the team plans ways to improve product quality by adapting the definition of done as appropriate.

Roadmap: The roadmap is the second level of Agile planning and distills the vision into a high-level plan that outlines work spanning one or more releases. Roadmaps are also used in traditional development programs, but typically have different names (see Rosetta Stone).

Scrum: Scrum is a framework for developing and sustaining complex products. See Appendix A for a brief description of Scrum and other Agile methods.

Scrumban: Scrumban is a mix between Scrum and Kanban, which supposedly contains the best features of both methods.

Scrum Master: Scrum is accountable for removing impediments to the ability of the team to deliver the sprint goal/deliverables. The ScrumMaster is not the team leader but acts as a buffer between the team and any distracting influences. The ScrumMaster ensures that the Scrum process is used as intended. The ScrumMaster is the enforcer of rules. A key part of the ScrumMaster's role is to protect the team and keep them focused on the tasks at hand. The role has also been referred to as servant-leader to reinforce these dual perspectives.

Should Have: Those features that are not critical for the project and do not need to be delivered as part of the requirements. However, these features are higher priority than the Could Have or Nice to Have features and could significantly improve the capability of the development project. (See also: Must Have, Could Have, and Nice to Have.)

Solution: Products, systems or services delivered to the business sponsor that provide value and achieve goals.

Spike: Often referred to as 'research' stories. Spikes usually are technical based and intended to provide just enough information that the team can estimate the size of the story or determine next step. Spike stories should be time- boxed and have a clearly defined deliverable.

Sprint: (see: Iteration)

Sprint Execution: Often also referred to as Sprint Execution, the recurring phase within the project lifecycle in which the Agile team executes against the iteration backlog. The goal of this phase is to complete all iteration commitments by delivering working software within the defined time constraints of the iteration. Typically, the iteration execution phase begins with the iteration kickoff and culminates with the iteration review.

Sprint\Iteration Backlog: At the beginning of each sprint, the team has sprint planning with an end result being a backlog of work that the team commits to completing by the end of the sprint. These are the items that the team will deliver against throughout the duration of the sprint.

Sprint Planning\Grooming: A pre-sprint planning meeting attended by the core agile team; during the meeting the Product Owner describes the highest priority features, as well as known acceptance criteria to the team as described on the product backlog. The team then agrees on the number of features they can accomplish in the sprint and plans out the tasks required to achieve delivery of those features. The planning group works the features into User Stories and assigns additional Acceptance criteria to each story.

Sprint Review: Each Sprint is followed by a Sprint review. During this review the software developed in the previous Sprint is reviewed with the Product Owner (prior to the Demo) and if necessary new backlog items are added.

Sprint Zero: Sprint Zero is the time used to plan and set-up the foundation needed before a team can execute their first successful iteration. Sprint Zero deliverables may include: identify stakeholders, build the team, training, create high-level architecture diagrams, and set-up environments.

Stable Team: A team with core team members are that are fully (at least 80%) dedicated to the team and not allocated across others or pulled in and out of the team.

Stakeholder: Anyone with an interest in the project. Specifically, parties that may be affected by the decision made, or can influence the implementation of its decisions. Stakeholder engagement is a key part of corporate social responsibility and achieving the project's vision.

Story: A high-level requirement definition written in everyday or business language; it is a communication tool written by or for the customers to guide developers though it can also be written by developers to express non-functional requirements (security, performance, quality, etc.). Stories are not vehicles to capture complex system requirements on their own. Rather, full system requirements consist of a body of stories. An epic is a large story that will eventually be broken down into smaller stories that will be captured in the backlog. Stories are used in all levels of Agile planning and execution. It captures the "who", "what" and "why" of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard. (also called "User Story").

Story Board: A wall chart (or digital equivalent) with markers (cards, sticky notes, etc.) used to track stories' progress for each iteration. For example, the board may be divided into "to do", "in progress", "done", etc. and the movement of the markers across the board indicates a particular story's progress. One goal of the story board may also be to recognize the order and the dependencies of the stories in representing end-to-end functionality for the users.

Story Maps: A visual technique to prioritize stories by creating a "map" of users, their activities, and the stories needed to implement the functionality needed.

Story Points: A unit of measure for expressing the overall size of a user story, feature, or other piece of work in the backlog. The number of story points associated with a story represents the complexity of the story relative to other stories in the backlog. There is no set formula for estimating the size of a story, rather, a story-point estimate is an amalgamation of the amount of effort involved in developing the feature, the complexity of developing it, and the risk inherent in it.

Sustainable Pace: A management workload philosophy that is a part of the XP Agile method (see Appendix A for a brief description of the XP method). It refers to a manageable, constant workload so that team will not be overextended. Sustainable pace, is crucial when using velocity to estimate how much work a team can during an iteration.

Task: A user story can be broken down in to one or more tasks. Tasks are estimated daily in hours (or story points) remaining by the developer working on them.

Taskboard/Storyboard: A wall chart with cards and sticky notes that represents all the work for in a given sprint; the notes are moved across the board to show progress.

Team: The Team is responsible for delivering the product. A Team is typically made up of 5–9 people with cross-functional skills who do the actual work (analyze, design, develop, test, technical communication, document, etc.). It is recommended that the Team be self-organizing and self-led, but often work with some form of project or team management.

Technical debt: The obligation that a software organization incurs when it chooses a design or construction approach that is expedient in the short term but that increases complexity and is more costly in the long term.

Test Driven Development: A software development process that relies on the repetition of a very short development cycle. For example, first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.

Time-box: A time-box is a previously agreed period of time during which a person or a team works steadily towards completion of some goal. Rather than allow work to continue until the goal is reached, and evaluating the time taken, the time-box approach consists of stopping work when the time limit is reached and evaluating what was accomplished. For example, in Scrum, the daily scrum is a 15-minute time-boxed event. This means that the daily scrum should take up to, but no longer than, 15 minutes to complete. Time-boxed iterations are typically associated with Scrum and XP.

Unit Testing: A software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine whether they are fit for use. This is the smallest testable part of any project.

User Persona: Personas are a description of the typical users of a given software. A persona description should include:

Skills and background – E.g. professional or beginner computer user
Goals – E.g. what does the user expect from the product?

User Stories: A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it. A user story is one or more sentences in the everyday or business language of the end user that captures what the user wants to achieve. A user story is also a placeholder for conversation between the users and the team. The user stories should be written by or for the customers for a software project and are their main instrument to influence the development of the software. User stories could also be written by developers to express non-functional requirements (security, performance, quality, etc.)

A recommended format for Users Stories is as follows:

*As a (**user role**), I would like (**statement of need**), so that I can (**desired benefit**). Or for example:*

As an Agile student, I would like an Agile glossary, so that I can understand the meanings of words that I hear in training.

Velocity: Velocity measure the amount of work a team can deliver each iteration. Typically, this is measured as story points accomplished per iteration. For example, is a team completed 100 story points during an iteration, the velocity for the team would be 100. Velocity is a team-specific metric and should not be compared across teams as a measure of relative productivity.

Verification and Validation Testing: Independent procedures that are used together for checking that the project meets the requirements and specifications; that is, that it fulfills its intended purpose.

Vision: The highest level of Agile planning, the vision is strategic in nature and is infrequently changed.

WIP Limits: Limiting the Work-in-Progress (WIP) so that the team maintains focus on completing stories\tasks, maintaining quality, and delivering value.

Work-In-Progress (WIP): Also known as Work in Progress is any work that has been started but has yet to be completed.

XP: A software development methodology that is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles (timeboxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.