Agile Glossary

Agile Term	Definition
Acceptance Criteria	Those criteria by which a work item (user story) can be judged to have been successfully implemented and tested. A story is 'done' when all criteria pass testing; conversely, a story is not 'done' if any criteria fail testing. Acceptance Criteria are discreet testable features that relate to the Conditions of Satisfaction that describe a higher level of conditions that, when met, deliver business value.
Acceptance Testing	The process to validate that a work item (user story) meets the specified acceptance criteria. Ideally, the approach is to automate as much as possible with tools and techniques available commercially and in the open source community. The result/output of acceptance testing is a report that identifies the status of each acceptance criteria being tested and whether that item passes or fails.
Agile Development	Agile development is a conceptual framework and approach to software development based on principles in the Agile Manifesto. The term is an "umbrella" for a number of specific methodologies based on iterative development techniques where requirements and deliverables evolve through collaboration between self-organizing, cross-functional teams. The most popular agile methodologies include: extreme programming (XP), Scrum, Crystal, Dynamic Systems Development (DSDM), Lean Development, and Feature Driven Development (FDD).
Agile Manifesto	A Manifesto for Agile Software Development is an historical document authored in February of 2001 at a ski resort in Utah. The meeting was held to discuss different approaches to lightweight, responsive, adaptable software development. The Manifesto represents their combined best thinking. It comprises two parts: four value statements and twelve principles. Here are the four value statements of the 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."
Agile Methods	Some well-known agile software development methods include: • Agile modeling • Agile Unified Process (AUP)

	• Dynamic Systems Development Method (DSDM) • Essential Unified Process (EssUP) • Feature Driven Development (FDD) • Open Unified Process (Open UP) • Scrum • Velocity Tracking
Agile Modeling	Agile Modeling is a practice-based methodology for Modeling and documentation of software-based systems. It is intended to be a collection of values, principles, and practices for Modeling software that can be applied on a software development project in a more flexible manner than traditional Modeling methods.
Agile Practices	Agile practices are procedures that are defined as being highly efficient to productivity, and include the following practices: user stories, cross-functional teams, unit testing, refactoring, continuous integration, multi-stage continuous integration, planning poker, burnup charts, burndown charts.
Application Lifecycle Management (ALM)	Also called ALM, Application Lifecycle Management is the management platform of the entire software application lifecycle, from planning to the final release. Key components of the platform include the ability to handle change management, workflow, source code management, task management, testing and bug tracking, reporting and analytics.
Backlog	Also knows as "product backlog," the backlog is a prioritized list of user stories and defects in order from most valuable to least valuable for a system. Backlogs include both functional and non-functional user stories as well as technical team-generated stories.
Behavior Driven Development (BDD)	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. It was originally named in 2003 by Dan North as a response to test-driven development, including acceptance test or customer test driven development practices as found in extreme programming. It has evolved over the last few years. 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. Behavior-driven developers use their native language in combination with the ubiquitous language of domain-driven design to describe the purpose and benefit of their code. This allows the developers to focus on why the code should be created, rather than the technical details, and minimizes translation between the technical language in which the code is written and the domain language spoken by the business, users, stakeholders, project management, etc.
Branching	Branching is the duplication of objects under revision control (such as a source code file, or a directory tree) in such a way that the newly created objects initially have the same content as the original, but can evolve independently of the original. Branching can take two forms, static or dynamic. In static branches, copy and label operations are used to duplicate a given branch. The duplicate then can evolve independently. With dynamic branches, usually implemented in

	streams, only the label operation is used, to flag the point in time that a stream diverged from its parent stream. Both branching forms support some form of merging, so that code changes made on a branch can be re-integrated into another branch, as is typical in parallel development processes.
Burndown Chart	Representation of the number of hours remaining for completion of a project; usually represented in chart form with points plotted on an x and y axis that map a downward trend of work left to do until burning down to zero.
Burnup Chart	Representation of the number of stories completed; usually represented in chart form with points plotted on an x and y axis that map an upward trend of work completed until reaching 100%.
Collocation (Collocated Teams)	Collocation refers to development teams located and working in the same location. Collocation is usually applied at the cross-functional team level. Collocation is an important (but not required) concept in Agile development to promote collaboration and osmotic communication.
Conditions of Satisfaction	High-level criteria by which a work item (user story) can be judged to have been successfully implemented and tested to deliver business value. A story is 'done' when all conditions pass testing; conversely, a story is not 'done' if any conditions fail testing. Conditions of Satisfaction are proved to be met by delivery of working code through Acceptance Testing.
Constraint	Anything that limits a progress from achieving higher performance; i.e., throughput, cycle time, quality, technical capabilities, etc.
Continuous Integration	Continuous integration, one of the foundational aspects of Agile software development methodologies, is defined by Martin Fowler to be "a fully automated and reproducible build, including testing, that runs many times a day. This allows each developer to integrate daily, thus reducing integration problems." By getting changes into the main line as frequently as possible, preferably daily, and by extending the idea of a nightly build, continuous integration helps reduce integrations problems and identify and resolve problems more quickly.
Cross-functional Team	Team comprised of members with all functional skills and specialties necessary to complete a project from start to finish.
Distributed Teams	Development teams that work on the same project but are located across multiple locations or worksites.
DMAIC	The five stages of a cycle of continuous improvement associated with the "six sigma" approach. ○ **Define** - Identify the stakeholders, the problem, and its scope. ○ **Measure** - Establish the metrics for analyzing the problem and determining the impact of any proposed changes. ○ **Analyze** - Review the collected data, establish performance gaps and variation, identify best

	practices ○ **Improve** - Design and develop a solution, validate and then implement the solution. ○ **Control** - Establish new standards, update measurement systems, and plan to maintain and improve.
Enterprise Agile Development	The adoption of specific Agile practices in an organization that works in conjunction with other non-Agile practices. Enterprise Agile Development is a highly efficient and customized practice for large organizations that have difficulty making a complete transition to Agile, as well as for organizations that already practice efficient development processes.
Epic	A user story which describes a large amount of customer value and needs to be broken down into many smaller user stories.
Feature-Driven Development (FDD)	Feature Driven Development (FDD) is an Agile method for developing software based on an iterative and incremental software development process. The main purpose of FDD is to deliver tangible, working software repeatedly in a timely manner.
Four D's	The four types of tasks that can make up an Agile story. ○ **Discover** - Gather/research information, define sprint deliverables. ○ **Develop** - Create implementation plans, construct and test deliverables. ○ **Deliver** - Train employees, execute plans, release deliverables. ○ **Debrief** - Monitor results, adjust deliverables, closeout stories.
Hybrid Development Processes	Development process that uses both Agile and non-Agile practices in conjunction with each other and is proven highly effective for some development teams
Information Radiator	A visual display of information about an Agile project that can easily be consumed without the need to interrupt anyone for status. Related primarily to Kanban.
Inspect and Adapt	An Agile concept where teams evaluate a project by looking at the product, listening to each other's feedback and ultimately improving the process or changing course.
Iron Triangle	A concept of project management to visualize a situation where all three project constraints of cost, scope, and time are fixed at the start of the project. As the project progresses, no one constraint may change without a change in at least one of the remaining two.
Iteration	A period of time in which software development activities take place and result in delivery of working software. Traditionally lasting between 2 and 4 weeks, iterations may be as short as 1 week or as long as 3 months. Similar in nature and definition as a "Sprint."
Kanban	A conceptual approach to Agile development based on Lean Software Development principles and has three main components: visual system for managing work, limits work in progress, and work is pulled

	rather than pushed through the system.
Lean	An umbrella term for a powerful combination of techniques to maximize customer value while minimizing waste and achieving continuous flow through a sustainable culture of continuous improvement. Lean is a term used in the U.S. for what was originally created as the "Toyota Production System". Also referred to as Lean Office, Lean Production, Lean Thinking, Lean Enterprise, etc.
Lean Software Development	A programming concept that focuses on optimizing efficiencies for development and minimizing waste. According to Mary Poppendieck, 10 rules of Lean programming include: eliminate waste, minimize artifacts, satisfy all stakeholders, deliver as fast as possible, decide as late as possible, decide as low as possible, deploy comprehensive testing, learn by experimentation, measure business impact and optimize across organizations.
Pair Programming	Process in which two developers work together at a single workstation, where one is responsible for typing code and the other for reviewing each line of code as it is typed in.
Parallel Development	Parallel development occurs whenever a software development project requires separate development efforts on related code bases. For example, when a software product is shipped to customers, a product development team may begin working on a new major feature release of the product, while a product maintenance team may work on defect corrections and customer patch releases of the shipped product. Both teams begin work from the same code base, but the code necessarily diverges. Frequently the code bases used in parallel development efforts must be merged at some future date, for example, to ensure that the defect corrections provided by the product maintenance team are integrated into the major release that the product development team is working on.
PDCA	The four stages of a cycle of continuous improvement popularized by W. Edward Deming. ○ **Plan** - Define the problem, methods to measure it, and obtain management support for future stages. ○ **Do** - Do the tests and prototypes to understand the problem, establish root causes, and investigate alternatives. ○ **Check** - Analyze the results of the "do" stage to determine if a solution effectively resolves the problem while breaking nothing else. ○ **Act** - Fully implement the identified solution.
Planning Poker	A consensus-based technique for estimating; mostly used to estimate effort or relative size of tasks in software development. Planning Poker is useful for building team cohesion and for fostering self-organizing teams.
Product Backlog	A prioritized list of user stories and defects in order from most valuable to least valuable for a system. Backlogs include both functional and non-functional user stories as well as technical team-generated stories. Owned by the Product Owner.
Product Owner	A role originating from Scrum, but has now been widely adopted

	independently of Scrum. A product owner manages the product backlog, addresses questions that arise during development and signs off on work results. The product owner guides the team with what should be done and when the final product should be shipped. The Scrum team then balances out the product owner's decisions by deciding how much work should be involved in an individual sprint and estimating the amount of time necessary to complete the task.
Refactoring	The practice of continuously improving the usability, maintainability, and adaptability of code without changing its behavior. Refactoring makes it much easier to add new and unanticipated functionality. Refactoring has the disadvantage that it takes extra effort and requires changing the code. Refactoring is, at times, used as a method of reducing technical debt.
Release	An increment of potentially shippable product from the development team into routine use by customers. Releases typically happen when one or more sprints has resulted in the product having enough value to outweigh the cost to deploy it. A release balances functionality, cost, and quality requirements against date commitments.
Release Management	Release management comprises a broad set of activities in software development organizations that center on ensuring that software is ready to be released to customers. Release management generally entails defining the functionality required to release to customers, a target date at which a release will be made and success criteria related to determining if the product is ready to release.
Release Plan	A document describing scheduling, activities, resources and responsibilities related to a particular release.
Retrospective	A meeting held at the end of every sprint to reflect on what went well during the sprint and what can be improved upon during the next sprint. Sprint retrospectives are valued as necessary parts of inspecting and adapting, and allow development teams to plan for future output.
Sashimi	An Agile concept of delivering value in slices rather than in layers/stages. An Agile Story is sashimi because it can be proven it is done. It is not possible to prove that a requirements document is done.
Scrum	Agile development project management framework based around sprints and is generally comprised of a Scrum Team, Product Owner and Scrum Master. The framework of Scrum leaves most development decisions up to the self-organizing Scrum team, where project decisions are reached by team consensus. Scrum was originally developed to guide complex software projects. Scrum delivers incremental iterative time-boxed sprints that deliver value early and often.
Scrum Master	Person trained to facilitate daily Scrum meetings, remove impediments, oversee the team's progress through the process and track Scrum team updates. The Scrum Master is responsible to enforce the principles and rules of engagement that a Scrum Team has agreed to put in place.

Self-Organizing	A team, usually found in Scrum, that manages itself through various means of communication and reoccurring structured meetings. Self-organizing teams solve development issues together as a whole and decide the best solution based on input, abilities and experience of the various team members.
Spike	Timeboxed investigation of feasibility via a basic implementation experiment or prototype to test out new, unknown, risky or complex technical solutions. The result of a Spike is to inform future implementation decisions.
Sprint	A Scrum-specific term used to describe an Iteration; although the term is widely used in other Agile approaches.
Sprint Backlog	The plan for a development team that maps out the activities, tasks and labor estimates to implement the User Stories planned for completion in an upcoming sprint.
Sprint Planning	A meeting for Scrum Teams, Scrum Masters and Product Owners where the Product Owner describes priority features to the team. The Scrum Team gets enough of an understanding about the tasks discussed that they are able to choose which ones to move from the product backlog to the sprint backlog.
Sprint Review	In the sprint review, teams go over what stories were completed during the iteration and demonstrate those stories for stakeholders and the product owner.
Stakeholder	Any party that has an interest in the product/service produced by an organization's value stream.
Stakeholder Value	The worth of the stakeholder's interest in the product/service produced by an organization's value stream. Sometimes used interchangeably with customer value.
Stand-up	Daily Meetings that are meant to quickly and efficiently resolve obstacles that any team members may be experiencing. A Stand-up meeting is no more than 15 minutes, focused on answering the questions: “What did you do since we last met? What will do before we meet again? And what obstacles are or may block your ability to meet that commitment?” Specific discussions that need to take place after the Stand-up meeting between team members are identified but tabled for follow-up discussion after the Stand-up.
Story Points	Relative scale of effort required by a team to implement a user story. A method for estimating the size of an Agile story used as an alternative to estimating the story in hours. Story points compare one story to another to determine a relative size and then assign points denoting that size.
Task	A discrete unit of work. Agile stories are broken down into a collection of tasks that must be performed to complete the story in one sprint. Tasks are typically sized to represent 4 to 8 hours of effort.

Task Board	A physical or electronic board representing the state of tasks in a current sprint, often divided into "to do," "in progress" and "done."
Test Driven Development (TDD)	An approach to development which combines “test-first” development practices where you write a test before you write just enough production code to fulfill that test and refactoring.
Theme	A collection of Agile stories that have a common affinity. A theme may contain several Epics and many User Stories.
Timeboxing	The practice of constraining the amount of time for performing any activity. Examples include iterations, spikes and stand up meetings.
Unit Testing	Tests that exercise small amounts of isolated functionality intended to validate that the written code meets basic acceptance tests.
User Stories	Used with Agile methodologies for specifying requirements and presented as an informal statement of the requirement in natural, simple-to-understand language. Typically stated in a format similar to: ○ As a <user or role>, ○ I want <business functionality>, ○ So that <business benefit>. User Stories also detail Conditions of Satisfaction (defining “done”) and Acceptance Criteria (defining “done right”)
Value Points	Relative scale of business value expected to be delivered by a User Story or feature.
Velocity	The velocity of a team is the number of story points associated with stories that are finished over a given period of time(typically an Iteration or Sprint). For instance, if the team completed 8 stories that were each 5 points during a four week sprint, then their velocity is 40 story points every four weeks.
Waterfall	Model of a software development process in which progress flows downwards through phases of conception, initiation, analysis, design, construction, testing and maintenance. Similar phases may be defined as define requirements, design the solution, develop, test, and implement.
XP (Extreme Programming)	"Extreme Programming," one implementation of the Agile methodology that focuses on producing the simplest coding situation for application requirements and includes practices such as pair programming, incremental design and continuous integration.