ASPE
RESOURCE
SERIES

The Agile Glossary of Terms

The skills we teach drive real project success.

Agile Glossary

Words and terms common to Agile methods

Acceptance Criteria

The specific criteria identified by the customer for each functional requirement. The acceptance criteria, written in easily to understand terms and from the customer's perspective, provides additional detail into how a feature should work and assesses the ability of the feature to perform its intended function.

Actual Time Estimation

A time-based method of estimating development work. The intent of this method is to best approximate the amount of time required to complete a given development task. Generally, these estimates are calculated using Ideal Engineering Hours.

> ***Examples:***
>
> *This task will be complete in 10 days. Or...*
> *This task will be complete by January 10th. Or...*
> *This task will require 25 development hours for completion.*

Agile

Often used as the abbreviation for Agile Software Development or Agile Methods. Agile is a generic term which refers to a collection of lightweight software development methodologies that value and support evolving requirements through iterative development, direct Customer/Developer communication and collaboration, self organizing cross-functional teams and continuous improvement through frequent inspection and adaptation.

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.*

AgileUP

Short for Agile Unified Process, it refers to a simplified version of the IBM Rational Unified Process (RUP). Much like RUP, AgileUP is an iterative process, that utilizes several Agile techniques and ideas for the purpose of developing business applications. The most noticeable difference between the two processes is that AgileUP only defines 7 key disciplines to RUP's 9. Those disciplines are:

1. **Model:** The goal of this discipline is to understand the business of the organization, the problem domain being addressed by the project, and to identify a viable solution to address the problem domain.

AgileUP
(cont.)

2. **Implementation:** The goal of this discipline is to transform your model(s) into executable code and to perform a basic level of testing, in particular unit testing.
3. **Test:** The goal of this discipline is to perform an objective evaluation to ensure quality. This includes finding defects, validating that the system works as designed, and verifying that the requirements are met.
4. **Deployment:** The goal of this discipline is to plan for the delivery of the system and to execute the plan to make the system available to end users.
5. **Configuration Management:** The goal of this discipline is to manage access to your project artifacts. This includes not only tracking artifact versions over time but also controlling and managing changes to them.
6. **Project Management:** The goal of this discipline is to direct the activities that take place on the project. This includes managing risks, directing people (assigning tasks, tracking progress, etc.) and coordinating with people and systems outside the scope of the project to be sure that it is delivered on time and within budget
7. **Environment:** The goal of this discipline is to support the rest of the effort by ensuring that the proper process, guidance (standards and guidelines), and tools (hardware, software, etc.) are available for the team as needed.

Application Lifecycle Management

Often abbreviated as ALM, it is generally used in reference to tools that help facilitate the coordination of business management and software engineering practices by integrating features related to requirements management, architecture management, coding, testing, defect tracking and release management into a single solution.

Burn Down Chart

A burn down chart is a simple, easy to understand graphical representation of "Work Remaining" versus "Time Remaining". Generally, "Work Remaining" will be represented on the vertical axis while "Time Remaining" is displayed along the horizontal axis. Burn down charts are effective tools for communicating progress and predicting when work will be completed. The burn down chart is also an effective means for teams to make adjustments in order to meet product/project delivery expectations.

Cadence

Cadence, by definition is a noun that represents the flow or rhythm of events and the pattern in which something is experienced. In verb form, it is used to describe the idea of making something rhythmical. Cadence is something that Agile teams strive to achieve as it allows teams to operate efficiently and sustainably within the iterative cycles that most Agile methods promote. In its simplest form, cadence allows Agile teams to focus on development and delivery of the product rather than on process.

Capacity

The measurement of how much work can be completed within a given, fixed time frame by estimating the number of available, productive work hours for an individual or team. To accurately estimate capacity, it is important to factor in all known variables such as meetings, holidays and vacations, as well as the effects of multi-tasking and normal administrative tasks.

Chickens and Pigs

From the popular Chickens and Pigs story by Ken Schwaber(see below). A "Chicken" is used to describe someone who, while involved in the process or project, is not committed and accountable for any specific deliverables. Chickens are often interested stake holders, managers and executives. As these individuals are not directly involved or accountable, it is encouraged that Chickens participation in the process is limited observation only. A "Pig", however, is an individual who is committed as they are directly accountable for specific project and product deliverables. Pigs are encouraged to wholly participate in the process as they will be accountable for the expectations set by their involvement and estimates.

The Chicken and Pig Story

A pig and a chicken are walking down a road. The chicken looks at the pig and says, "Hey, why don't we open a restaurant?" The pig looks back at the chicken and says, "Good idea, what do you want to call it?" The chicken thinks about it and says, "Why don't we call it 'Ham and Eggs'?" "I don't think so," says the pig, "I'd be committed, but you'd only be involved."

Complexity Points

Complexity points are units of measure, based on relative sizing, used to estimate development work in terms of complexity and/or size, versus traditional time based methods which attempt to measure the duration of time required to complete some unit of work. Complexity Points are similar to 'Story Points' (see Story Points) but the scale used for complexity points may vary based on the differences in implementation of this sizing approach.

Continuous Integration

The practice of continuously integrating new development code into the existing codebase. Continuous integration allows the development team to ensure that the code repository always reflects the latest working build of the software. As developers complete the coding of a feature, the feature is applied to the latest software build where it is validated for defects and integrated into the codebase previously delivered. Continuous integration practices generally include testing and build automation, resulting in an end-to-end integration suite.

Crystal

Crystal (sometimes referred to as Crystal Clear) is a lightweight, Agile software development framework developed originally by Alistair Cockburn. Crystal, as a basic matter of principle, is primarily focused on the collaboration and interactions between teams of people rather than the processes and artifacts of traditional methodologies. Crystal methods value:

- Frequent delivery of working software
- Continuous, reflective improvement
- Osmotic communication via team colocation
- Technical excellence by utilizing automated testing, configuration management and frequent integration

Customer Unit	The Customer Unit refers to the people and roles that define and/or represent the voice and expectations of the primary consumers of the deliverables produced throughout the course of the project. Product Managers, Sales, Marketing, Executives and End-User Customers are all examples of roles and titles that often comprise the Customer Unit. In Agile projects, the Customer Unit is typically responsible for setting the vision, project charter and roadmap, creating and maintaining product backlog requirements and priorities, defining user acceptance criteria and communicating with the Developer Unit.
Daily Scrum	The Daily Scrum, also referred to as 'the daily stand-up', is a brief, daily communication and planning forum, in which Agile/Scrum teams come together to evaluate the health and progress of the iteration/sprint. It is also considered to be the fifth and final level of the Agile planning process. As a daily, team planning meeting, an effective daily scrum should be a tightly focused and time boxed meeting that occurs at the same time and place, on a daily basis. The intent of the daily scrum is to better understand the progress of the iteration, by all contributing team members honestly answering the following three questions: 1. What did I accomplish yesterday? 2. What will I commit to, or complete, today? 3. What impediments or obstacles are preventing me from meeting my commitments? Conversation in these meetings should remain focused on answering these three questions only. For the sake of brevity and greater team efficiency, additional discussion stemming from these three questions should be handled independently of the daily scrum, and should be limited to those team members who are directly involved.
Demo (Demonstration)	At the end of each iteration, the development unit performs a demo of the functionality completed during the iteration. The demo is a forum for the customer to provide feedback on the product's development to influence the evolution of the product.
Developer Unit	The Developer Unit refers to the people that are responsible for delivering working software that meets requirements by collaborating with the customer throughout the development lifecycle. Typically, the Development Unit is comprised of individuals fulfilling the technical roles of development (developers, QA, tech writer, project manager, DBA and others), working together in one or more, cross-functional agile teams. In agile projects, the developer unit is responsible for estimating the backlog, working with the customer unit to plan the iterations, iteration execution, demonstration and ultimate delivery of working software.

Dynamic Systems Development Method

Often abbreviated as DSDM, it is another example of a lightweight, Agile software development methodology. DSDM is an iterative and incremental approach that is largely based on the Rapid Application Development (RAD) methodology. The method provides a 4 staged/phased framework consisting of:

1. **Feasibility & Business Study**
2. **Functional Model / Prototype Iteration**
3. **Design and Build Iteration**
4. **Implementation**

Within each of the different phases, DSDM relies on several different activities and techniques that are all based on the following key, underlying principles:

- Projects best evolve through direct and co-located collaboration between the developers and the users.
- Self managed and empowered teams must have the authority to make time sensitive and critical project level decisions.
- Design and Development is incremental and evolutionary in nature and is largely driven by regular and iterative user feedback.
- Working software deliverables are defined as systems that address the critical and current business needs versus systems that address less critical and future needs.
- Frequent and incremental delivery of working software is valued over infrequent delivery of perfectly working software.
- All changes introduced during development must be reversible.
- Continuous integration and QA Testing is conducted in-line, throughout the project lifecycle.
- Visibility and transparency is encouraged through regular communication and collaboration amongst all project stakeholders.

Epic Stories

Epic stories are user stories whose scope is so large as to make them difficult to complete in a single iteration or accurately estimate the level of effort to deliver. Epic stories, while common when first defining the product backlog (see product backlog), should be decomposed into smaller user stories where the requirements of the story are defined much more narrowly in scope.

---

eXtreme Programing

Often abbreviated as XP, it is a popular example of a lightweight, Agile software development method. XP seeks to improve software quality by focusing on technical excellence, while improving project agility and responsiveness to changing requirements by valuing small yet frequent, time-boxed releases. XP provides a basic framework for managing projects based on the following key values:

- **Communication:** The most effective way to communicate requirements is by direct communication between the user and the developer
- **Simplicity:** Focus on building the simplest solution that meets the needs of today.
- **Feedback:** Inspect, adapt and evolve the system by responding to feedback from system tests, user acceptance tests and team input.
- **Courage:** By having the courage to refactor in the future, we can focus on only building what we need today.
- **Respect:** Do no harm to others by striving for the highest degree of quality in the solutions you build today.

Feature Based Planning

Feature based planning is an approach used in release planning, where features and scope take priority over date. Release plans (see Release Planning) created utilizing this approach are created by estimating the amount of time that will be required to complete a certain defined amount of scope. Often this approach is utilized for new product launches where a minimal or critical amount of feature/function must be delivered in order for the completed product to be considered market worthy.

Feature Driven Development

Feature driven development, or FDD, is an example of a lightweight, Agile approach to development. Like many other Agile methods, FDD seeks to deliver valuable, working software frequently in an iterative manner. FDD utilizes an incremental, model driven approach that is based on the following 5 key activities:

1. **Develop the Overall Model:** Define the high level technical and functional architecture and project vision.
2. **Build the Feature List:** Define the individual features that are required to meet the needs of the defined model.
3. **Plan by Feature:** Create the actual working development plans by grouping the defined requirements into feature groups or themes.
4. **Design by Feature:** Based on the feature based plans, design packages are created that seek to group together small feature sets that can be developed within 2 week, iterative periods.
5. **Develop by Feature:** Development feature teams, execute against the design packages and promote completed and tested code to the main build branch within the 2 week, time-boxed iteration.

Feature Teams

Feature teams are small, cross-functional teams of development resources, focused on designing and building specific feature groupings or areas of system functionality. The feature team concept is utilized by FDD and other agile methods.

Fibonacci Sequence

Discovered in the 12th century by Leonardo Pisano, the Fibonacci sequence is a mathematically recursive sequence, in which the result of each subsequent term is determined by the sum of the two previous terms. A classic example of this concept is illustrated in the following string of numbers:

1, 1, 2, 3, 5, 8, 13, 21...

Using this example above, 1+1=2, 1+2=3, 2+3=5 and so on. The Fibonacci sequence serves as the basis of popular agile estimating technique known as Planning Poker.

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.

Ideal Hours

Ideal hours is a concept often used when applying time-based estimates to development work items. Ideal time is the time it would take to complete a given task assuming zero interruptions or unplanned problems. Many time-based estimation methods utilize this time scale when planning and estimating. Considering the grossly optimistic assumptions this approach takes, the accuracy of ideal estimates are often inversely proportional to the duration of the estimate.

INVEST (acronym)

Coined by Bill Wake in eXtreme Programing Explored, 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

Often also referred to as a Sprint, an iteration is a predefined, time-boxed and recurring period of time in which working software is created. The most commonly used iteration durations are 2, 4 and 6 week periods. The iteration level is also considered to be the fourth level in the five level Agile planning process.

Note: The terms Sprint and Iteration are synonyms and are effectively interchangeable. The term sprint is widely used by teams that identify their Agile approach as Scrum, whereas iteration is a more generic term used in the same manner.

Iteration Backlog

Often also referred to as the Sprint Backlog, the iteration backlog is a subset of user stories from the product backlog,that contains the planned scope of a specific iteration. Generally, the iteration backlog reflects the priority and order of the release plan and product roadmap.

Iteration 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.

Iteration Plan

Often also referred to as the Sprint Plan, the iteration plan is the detailed execution plan for a given (usually current) iteration. It defines the iteration goals and commitments by specifying the user stories, work tasks, priorities and team member work assignments required to complete the iteration. The iteration plan is normally produced by the entire development unit during the iteration planning session.

Iteration Review

Often also referred to as Sprint Review, the iteration review is an important communication forum that occurs at the end of an iteration. During the iteration review an Agile team will evaluate and agree on which stories have been completed and which stories need to be deferred or split. The iteration review is an event that generally signifies the closing of an iteration.

Kano Analysis

Developed by Professor Noriako Kano, it is a method used for classifying and categorizing requirements (user stories) based on their impact to customer satisfaction. The Kano Analysis model utilizes four categories into which each requirement can be classified. Those categories are:

- **Must Have/Must Be:** Baseline features, functional barriers to entry. Without these features customers won't use the product.
- **Satisfiers:** These are the features that a customer will generally expect and make the difference between a satisfying user experience versus one that is simply adequate. The more satisfiers the better.
- **Exciters and Delighters:** These are the features that 'delight' your customers, they love your product because of these features. Generally these are product differentiators.
- **Dissatisfiers:** These are features that customers do not want and should not be delivered. Dissatisfiers emerge as a backlog ages and better ideas are identified and delivered.

Lean Software Development

Lean Software Development, which is rooted in the Lean Manufacturing techniques developed by Toyota, is another popular example of a lightweight Agile approach to product development. Much like other Agile methods, Lean attempts to address the shortcomings of traditional software project management methods by focusing on people and effective communication. Lean is further defined by the following seven key principles:

- **Eliminate Waste:** Understand your customers needs and seek to deliver solutions that address only those needs as simply as possible.
- **Create Knowledge:** Create a team-based environment in which all individual participate in the design and problem-solving process. Create a culture that encourages constant improvement through regular inspection and adaptation.
- **Build Quality In:** Embrace re-factoring and test automation and test driven development. Understand your test requirements and plans before you begin coding.
- **Defer Commitment:** Avoid dependencies by embracing loose coupling. Maximize flexibility by narrowly defining requirements and schedule irreversible decisions to the last possible moment.

Lean Software Development
(cont.)

- **Optimize the Whole:** Focus on the entire value stream. Understand that a completed product is the sum of the all the various contributors and the result of effective collaboration.
- **Deliver Fast:** Reduce risk and volatility by limiting scope. Commit only to work for which you have the capacity to complete. Decompose work to discrete actionable tasks and release work often and iteratively.
- **Respect People:** Trust that your people know best how to do their jobs and empower them to make the decisions needed to complete their commitments.

Meta-Scrum

The Meta-Scrum is a communication forum that is often used in larger projects that scale across multiple Agile teams, for the purpose of coordinating resources and dependencies. Generally, this planning forum will involve the product owners, project managers and scrum masters.

MoSCoW

MoSCoW is a feature classification/categorization method, rooted in rapid application development, that is commonly utilized in Agile projects. The method is intended for short, time-boxed development iterations where focus should remain on those items that are deemed most critical for delivery within the time-boxed period. MoSCoW itself is a modified acronym, which represents 4 different levels of priority classification.

- **Must Have:** These are time critical project requirements that must be delivered in order for the project not to be considered an outright failure. These are generally baseline, or critical path features.
- **Should Have:** These are also critical project level requirements, however they are not as time critical as Must Have requirements.
- **Could Have:** These are considered to be the Nice to Have requirements. Features that are not necessarily required for the success of the iteration or project, but features that would increase end-user/customer satisfaction in the completed product.
- **Won't Have:** These are lowest priority requirements that will not be scheduled or planned within the delivery time box.

Paired Programming

A programming technique where two developers work together at a single workstation on the development of a single feature. Paired programming allows for better review of the code as it is created and also allows one developer to consider the strategic direction of the feature while the other produces the code. Paired programming often yields higher quality code and can allow for cross-training opportunities among team members of differing skill levels.

Persona

A fictional character that is created to represent the attributes of a group of the product's users. Personas are helpful tools to use as a guide when deciding on a product's features, functionality, or visual design. Personas allow a team to easily identify with a fictional version of the product's end users.

Planning Game

A planning meeting with the goal of selecting user stories for a release or iteration. The user stories selected for inclusion in the iteration or release should be is based on which user stories will deliver the highest value to the business given current development estimates.

Planning Poker

Planning poker is a team based exercise that is commonly used for assigning relative estimate values to user stories/requirements to express the effort required to deliver specific features or functionality. The game utilizes playing cards, printed with numbers based on a modified fibonacci sequence (0,1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100). Equipped with playing cards, all members of the development unit team and the product owner meet together to discuss product backlog requirements for the purpose of reaching a consensus based estimate. The rules of the game are as follows:

- The team and the product owner select from the backlog a requirement that all agree is small enough in size and complexity to be considered a 2. This becomes the baseline requirement.
- With the baseline requirements selected, the product owner selects a requirement from the backlog and describes it greater detail, allowing the team to ask any clarifying questions. Discussion regarding the requirement should be limited in time to 5–8 minutes.
- Once everyone on the team is satisfied with the details, each player pulls a numbered card from their deck that they feel best represents the current requirements size and complexity relative to the baseline requirement or other known, estimated requirements. This card is placed face down.
- Once all the players have made their selection, all cards are turned face up and revealed to the group.
- Those players with high and/or low estimates are asked to justify their selection by discussing with the team.
- After the outliers have justified their selections, the team repeats the estimation process until group consensus achieved.
- This process is repeated as much as needed in order to score all requirements from the backlog.

Product Backlog

The product backlog is a prioritized and estimated list of all outstanding product/project requirements, features, defects and other work items. The product backlog is typically owned and managed by the product owner who reviews it on a regular cadence to ensure that the development unit is focusing on the completion of those items that represent the highest impact on the overall product value.

Product Owner

Often referred to as the "Voice of Customer" on Agile projects or the "Product Manager" on traditional projects, the product owner is the person responsible for communicating the customer requirements. While considered part of the customer unit, the product owner role is critical to the success of an Agile product development effort. In addition to communicating requirements, the product owner is responsible for defining user acceptance criteria for requirements, prioritizing and managing the product backlog, as well as collaborating with the development unit throughout the iterative development cycle.

Rapid Application Development

Rapid application development, often abbreviated simply as RAD, is a software development methodology that favors rapid and iterative prototyping in lieu of detailed and comprehensive plans, and is often considered an example of Agile development methods. RAD promotes a highly collaborative, team based approach to developing software, by evolving requirements through frequent and iterative delivery of working prototypes.

Rational Unified Process

Created by Rational Software (which was later acquired by IBM), the Rational Unified Process (RUP) is an iterative development process that seeks to increase development agility by providing a flexible, best practice based life cycle management framework. RUP prescribes the utilization of 9 key disciplines extended across 4 main project phases. Those phases are:

1. **Inception Phase**
2. **Elaboration Phase**
3. **Construction Phase**
4. **Transition Phase**

The 9 key disciplines are as follows:

1. **Model:** The goal of this discipline is to understand the business of the organization, the problem domain being addressed by the project, and to identify a viable solution to address the problem domain.
2. **Requirements:** The goal of this discipline is to elicit stakeholder feature/ function requirements in order to define the scope of the project.
3. **Analysis and Design:** The goal of this discipline is the define the requirements into actionable and executable designs and models.
4. **Implementation** The goal of this discipline is to transform your model(s) into executable code and to perform a basic level of testing, in particular unit testing.
5. **Test** The goal of this discipline is to perform an objective evaluation to ensure quality. This includes finding defects, validating that the system works as designed and verifying that the requirements are met.
6. **Deployment** The goal of this discipline is to plan for the delivery of the system and to execute the plan to make the system available to end users.
7. **Configuration Management** The goal of this discipline is to manage access to your project artifacts. This includes not only tracking artifact versions over time but also controlling and managing changes to them.
8. **Project Management** The goal of this discipline is to direct the activities that takes place on the project. This includes managing risks, directing people (assigning tasks, tracking progress, etc.)and coordinating with people and systems outside the scope of the project to be sure that it is delivered on time and within budget.
9. **Environment** The goal of this discipline is to support the rest of the effort by ensuring that the proper process, guidance (standards and guidelines), and tools (hardware, software, etc.) are available for the team as needed.

Refactoring	Refactoring refers to the process of modifying and revising development code in order to improve performance, efficiency, readability, or simplicity without affecting functionality. As Agile methods advocate simplicity in design, solving only the problems that exist today and technical excellence through continuous improvement, code refactoring is something that should be embraced by teams and made part of the normal development process. Refactoring is often seen by management as an investment in the value of longevity and adaptability of the product over time.
Regression Test	A test completed to validate previously completed and tested code. The regression test is performed in an effort to ensure that subsequent deliveries of code segments have not corrupted previously completed code. These tests are also often performed after defects are remediated to ensure that the fixes have not corrupted any other portion of the software.
Relative Estimation	Relative estimation is a software estimation technique that attempts to size development requirements and work items not in terms of time or duration, but rather in terms of size and complexity relative to the size and complexity of other known requirements and work items. Relative estimation is commonly used in Agile development methods, and forms the basis of the planning poker estimation game.
Release Plan	A release plan is a document that further distills the roadmap by describing all of the anticipated activities, resources, and responsibilities related to a particular release, including the estimated duration of that release. Unlike in traditional waterfall managed projects, Agile methods seek to ensure the highest degree of plan accuracy by encouraging regular and iterative re-planning based on actual iteration results. Additionally, release planning is considered to be the third level in the five- level Agile planning process.
Retrospective	By dictionary definition, retrospective refers to process of looking back on, and/or contemplating the past. In Agile methods, a retrospective is a communication forum in which Agile teams come together to celebrate team successes and to reflect on what can be improved. The goal of the meeting is to develop a plan that the team will use to apply lessons learned going forward. Unlike in traditionally managed projects where these meetings (often called "post-mortems" or "lessons learned" meetings) are typically held at the conclusion of a project, Agile methods advocate a more regular and iterative approach, and encourage scheduling these meetings at the conclusion of each and every iteration. Retrospectives are an immensely powerful tool and are extremely useful in fostering an environment of continuous improvement.
Roadmap	The roadmap (or product roadmap) is a document that further distills the product vision as defined in the product charter into a high level plan. Commonly, the roadmap will attempt to outline project work that spans one or more releases, by grouping requirements into prioritized themes and estimating the execution schedule against said themes. Additionally, roadmap is considered to be the second level in the five-level Agile planning process.

Schedule Based Planning

Schedule based planning is an approach used in release planning, where date and schedule take priority over features and scope. Release plans created utilizing the approach are created by estimating the amount of scope that can be completed within defined release time box.

Scrum

Scrum is a incremental and iterative software development framework, and is arguably one of the most commonly used Agile methods. Unlike other Agile methods, Scrum is not an acronym. The method was coined as such by Hirotaka Takeuchi and Ikujiro Nonaka in a HBR article titled 'The New Product Development Game' written in 1986, in reference to the sport of rugby. The process involved was developed by Ken Schwaber and Dr. Jeff Sutherland.

Scrum outlines a process framework in which Product Owners, Scrum Masters and Team Members, all work together collaboratively to define product and sprint backlogs that are executed in short, time-boxed iterations that are called sprints. At the end of each sprint, a working increment of the software is delivered/demonstrated to the product owner and the entire process repeats itself.

ScrumMaster

A key role in the Scrum product development framework, the ScrumMaster is the person who is primarily responsible for facilitating all Scrum meetings, removing team impediments, protecting teams from external distractions, keeping the team honest and on-track to ensure that the team is best able to deliver against the sprint goals. As Scrum teams are self-organizing and self-managed, it is important to differentiate between a ScrumMaster and a traditional manager. Rather than attempt to manage the scrum team, effective ScrumMasters work for the team, and are often best described as servant-leaders. Although many ScrumMasters were once traditional project managers, ScrumMasters focus on achieving the best performance from the product team and holding the team accountable to their commitments.

Scrum of Scrums

Similar in intent to the Daily Scrum (or Daily Stand Up), the Scrum of Scrums is a daily communication forum commonly used in larger projects utilizing multiple scrum teams. As more teams are introduced, the likelihood of intra-team impediments due to overlapping work and dependencies increases. The Scrum of Scrums is an effective way of managing these impediments. Typically, this meeting occurs after all of the individual team Scrum meetings have been completed. An individual from each Scrum team (usually the Scrum Master) is tasked with representing the Scrum team in this meeting.

The agenda of this meeting is virtually identical to that of the daily scrum, other than the three questions are modified slightly to better reflect the focus on teams.

1. What did my team accomplish yesterday?
2. What will my team commit to, or complete, today?
3. What impediments or obstacles are preventing my team from meeting its commitments?

Single Done Definition	Single Done Definition is a concept that attempts to address the phenomena where the number of different interpretations of the word "done" is equally proportional to the number of individuals and roles found in an Agile team. Understanding that progress is measured in terms of delivered, working software, and that team credit for iteration commitments is only realized when said commitments are fully completed, it becomes clear that a common team understanding of "completed" is required. By explicitly defining and documenting all of the various elements that must be completed before a work item can be considered "done", the likelihood of delivering working software is improved as teams ensure a consistent and common understanding of "done" as it pertains to team iteration commitments.
Sprint	Often also referred to as an *Iteration*, a sprint is a predefined, time-boxed and recurring period of time in which working software is created. The most commonly used sprint durations are 2, 4 and 6 week periods. The sprint level is also considered to be the fourth level in the five-level Agile planning process. *Note: The terms* Sprint *and* Iteration *are synonyms and are effectively interchangeable. The term sprint is widely used by teams that identify their Agile approach as Scrum, whereas iteration is a more generic term used in the same manner.*
Sprint Backlog	Often also referred to as the *Iteration Backlog*, the sprint backlog is a subset of user stories from the product backlog, that contains the planned scope of a specific iteration. Generally, the iteration backlog reflects the priority and order of the release plan and product roadmap.
Sprint Plan	Often also referred to as the *Iteration Plan*, the sprint plan is the detailed execution plan for a given (usually current) iteration. It defines the iteration goals and commitments by specifying the user stories, work tasks, priorities and team member work assignments required to complete the iteration. The Sprint Plan is normally produced by the entire development unit during the sprint planning session.
Sprint Review	Often also referred to as *Iteration Review*, the sprint review is an important communication forum that occurs at the end of a sprint. During the sprint review an Agile team will evaluate and agree on which stories have been completed and which stories need to be deferred or split. The sprint review is an event that generally signifies the closing of a sprint. Often times, teams will also use the sprint review as a forum to demonstrate the work that was completed within the sprint.
Story Points	Story points are unit-less measures of relative size assigned to requirements for functionality. Story points are assigned by the entire team utilizing the planning poker exercise. Story points allow the team to focus on the pure size and complexity of delivering a specific piece of functionality rather than trying to perfectly estimate a duration of time required for the completion of the functionality.

Story Review	The story review forum, is a brief meeting that occurs prior to the start of new iteration and is attended by the product owner and development team. The intent of this meeting is to give teams a better understanding regarding upcoming stories/requirements so that they may be better prepared for the iteration planning meeting. Generally, story review meetings are between an hour or two in length and are scheduled about one week prior to the upcoming iteration planning/kickoff meeting.
Sustainable Pace	The concept that developers should not work an excessive number of hours due to the possibility of "developer burnout." This approach reflects studies that have determined the team productivity greatly decreases when teams work in excess of 40 hours per week. Teams working at a sustainable pace are more effective in accurately predicting their capacity over time while also maintaining the quality of their deliverables.
Task Boards	Task boards are visual communication and planning tools that are extremely useful for teams working in co-located environments. Typically, task boards take the form of a simple matrix, utilizing different work states (examples: not started, in progress, QA ready, completed) as column headers, and User Story Cards as the row headers. Within the matrix are the discrete tasks that describe the work required to complete a story. As the iteration progresses, tasks should move from one end of the task board to the other, through all of the various states. Due to their intuitive and simple nature, tasks boards provide a powerful means of measuring and communicating iteration health and progress.
Team Member	Simply put, a team member is anyone on a scrum team who is not a Product Owner or Scrum Master, who is responsible for specific iteration deliverables and is accountable for contributing to team iteration commitments and goals. A team member may be a developer, a technical writer, an architect, quality analyst or any other role essential to the production and delivery of working software.
Test Automation	The practice of using software to automate the testing process. Testing automation requires up-front planning and configuration of the testing software to ensure that the execution of the test meets the expectations of the customer. Test automation allows for more frequent regression testing without increasing the resource requirements to execute the tests.
Test Driven Development	More a technique than an actual development methodology, test driven development, often abbreviated as TDD, is a practice that is commonly utilized in Agile methods. TDD advocates the discipline of building working code, by first designing and building tests that exercise how the completed code should work, then creating the code so as to make all the pre-defined tests pass. The idea being that if a developer first understands the tests required to validate how the code should work and when it is complete, that developer will be much more thoughtful and successful

Test Driven Development *(cont.)*

in designing and developing working, quality software. While TDD promotes the utilization of automated testing tools combined with version and source code control tools, automation should not be considered a strict requirement as there is still considerable value in simply following the basic, quality driven principles of TDD.

Unit Test

A test performed by the developer to verify and validate the code that the developer completed is fit for use. The Unit Test is often the first level of testing that is completed as a part of a comprehensive test approach for software development.

Use Case

A use case is a document that attempts to describes system behavior from an end-user's perspective, by outlining the flow of data, system behavioral interchanges and corresponding end-user interactions in a sequential, step-by-step manner. While there is no single standard or format that they must follow, and use cases should vary in detail based on the needs of the requirements, uses cases often make use of the following artifacts to describe how a system should work.

- **Goal:** What is the end goal and desired effect of the functionality that the use case is attempting to describe.
- **Summary:** A brief, high-level and easily understood description of the use case.
- **Actors:** The consumers of the system that will be interacting with the system within the scope of the use cases. Actors can be people or other systems or services.
- **Preconditions:** System conditions and assumptions that must be true for the use case to be valid.
- **Triggers:** The specific events required to initiate the use case.
- **Body Text:** The description of each of the steps involved/required to complete the use case. Generally, body text will focus only the main (happy) path.
- **Alternative Path:** Steps that deviate from the main path due to exceptions, alternative logic or other conditional events.
- **Post Conditions:** The changes in the system and data as a result of executing steps outlined in the use case.

User Acceptance Tests

User acceptance tests describe the tests that must be successfully executed in order to validate that specific piece of functionality meets the needs of the user as outlined in the customer requirements. As many Agile methods advocate the use of narrowly defined requirements that speak from a specific user's perspective (i.e. user stories), it is recommended that user acceptance criteria follow similar form and define validation steps from the same user perspective. User acceptance tests are an essential component of user requirements, as without well defined acceptance criteria, it becomes difficult to clearly define the scope of any given requirement.

User Roles

A key ingredient in defining Agile requirements, user roles are used to describe the unique perspectives of the different consumers that will interact with the working software. Much like actors in a use case, user roles should not just be limited to the human consumers of the software and should also include any other relevant external software or service.

User Stories

User stories are simple, brief and concise statements, used to describe customer software requirements, from a particular user's perspective. User stories are often used in Agile methods for capturing and communicating customer requirements, as their short format allow for quick and easy requirements gathering, and their high-level structure encourages design evolution through team collaboration.

Commonly captured on 3x5 index cards so as to force brevity, user stories should seek to describe requirements as granularly as possible while still retaining customer value. User stories are not intended to describe complex system requirements on their own, but rather only narrow portions of requirements singularly, and through the whole, the full system requirements emerge.

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

Used in combination with relative (story point) estimation, static teams, and fixed time-boxed iterations, velocity is a predictive metric that is useful for long/mid term planning and estimation. Simply put, velocity attempts to measure the number of story points a single team can complete within a single, time-boxed iteration. Assuming that all of the requirements in a product backlog have all been estimated using a similar relative estimation point scale, it becomes possible to estimate the number of time-boxed iterations required to complete said backlog by simply dividing the team velocity value by the sum of the backlog's complexity point total.

Vision

Vision is the first and highest level in the Agile planning process. Activities that are often associated with this level of planning are the creation of project charters, feasibility studies, funding decisions, definition of project success factors/metrics, and the formation of individual teams. Typically, vision planning is very strategic in nature and only occurs on an annual basis.

Working Software

The term used to describe the level of completeness that features developed during an iteration should achieve by the conclusion of the iteration. Working software implies that the features demonstrated to the customer at the end of an iteration should be functionally complete and will not require redevelopment in order to prepare the functionality for a production environment.

> *Used in one of the 12 Agile Principles*
>
> *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*