Agile Resource Starter Pack

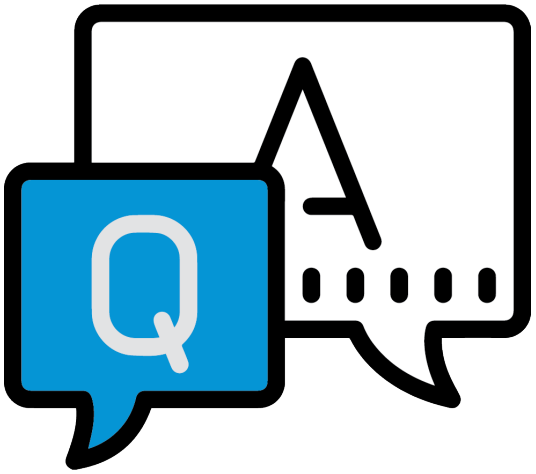
Scrum and Agile FAQ
Agile Encyclopedia
Agile Scrum Glossary
Scrum Team Cheat Sheet
Scrum Master Cheat Sheet
Product Owner Cheat Sheet

cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

What is Agile? What is Scrum?

The FAQ Guide for everything you need to know



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

Agile Defined:

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto. The Manifesto was developed by a group fourteen leading figures in the software industry, and reflects their experience of what approaches do and do not work for software development. Read more about the Agile Manifesto.

Scrum Defined:

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one.

- A "process framework" is a particular set of practiceSs that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.)
- "Lightweight" means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.

A Scrum process is distinguished from other agile processes by specific concepts and practices, divided into the three categories of Roles, Artifacts, and Time Boxes. These and other terms used in Scrum are defined below. Scrum is most often used to manage complex software and product development, using iterative and incremental practices. Scrum significantly increases productivity and reduces time to benefits relative to classic "waterfall" processes. Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements, and produce a product that meets evolving business goals. An agile Scrum process benefits the organization by helping it to

- Increase the quality of the deliverables
- Cope better with change (and expect the changes)
- Provide better estimates while spending less time creating them
- Be more in control of the project schedule and state

Benefits of Agile

Benefits to Customer

Customers find that the vendor is more responsive to development requests. High-value features are developed and delivered more quickly with short cycles, than with the longer cycles favored by classic "waterfall" processes.

Benefits to Vendors

Vendors reduce wastage by focusing development effort on high-value features, and reduce time-to-market relative to waterfall processes due to decreased overhead and increased efficiency. Improved customer satisfaction translates to better customer retention and more positive customer references.

Benefits to Development Teams

Team members enjoy development work, and like to see their work used and valued. Scrum benefits Team members by reducing non-productive work (e.g., writing specifications or other artifacts that no one uses), and giving them more time to do the work they enjoy. Team members also know their work is valued, because requirements are chosen to maximize value to customers.

Benefits to Product Managers

Product Managers, who typically fill the Product Owner role, are responsible for making customers happy by ensuring that development work is aligned with customer needs. Scrum makes this alignment easier by providing frequent opportunities to re-prioritize work, to ensure maximum delivery of value.

Benefits to Project Managers

Project Managers (and others) who fill the ScrumMaster role find that planning and tracking are easier and more concrete, compared to waterfall processes. The focus on task-level tracking, the use of Burndown Charts to display daily progress, and the Daily Scrum meetings, all together give the Project Manager tremendous awareness about the state of the project at all times. This awareness is key to monitoring the project, and to catching and addressing issues quickly.

Benefits to PMOs and C-Level Executives

Scrum provides high visibility into the state of a development project, on a daily basis. External stakeholders, such as C-Level executives and personnel in the Project Management Office, can use this visibility to plan more affectively, and adjust their strategies based on more hard information and less speculation.

Scrum Requirements

Scrum does not define just what form requirements are to take, but simply says that they are gathered into the Product Backlog, and referred to generically as "Product Backlog Items," or "PBIs" for short. Given the time-boxed nature of a Sprint, we can also infer that each set should require significantly less time to implement than the duration of the Sprint. Most Scrum projects borrow the "XP" (Extreme Programming) practice of describing a feature request as a "User Story," although a minority uses the older concept of a "Use Case." We will go with the majority view here, and describe three reasonably-standard requirements artifacts found in Product Backlogs.

User Story

A User Story describes a desired feature (functional requirement) in narrative form. User Stories are usually written by the Product Owner, and are the Product Owner's responsibility. The format is not standardized, but typically has a name, some descriptive text, references to external documents (such as screen shots), and information about how the implementation will be tested. For example, a Story might resemble the following:

Name: Planner enters new contact into address book, so that one can contact the person later by postal or electronic mail

Description: Planner enters standard contact information (first and last name, two street address lines, city, state, zip / postal code, country, etc.) into contact-entry screen. One clicks "Save" to keep the data, and "Cancel" to discard data and return to previous screen.

Screens and External Documents: http://myserver/screens/contact-entry.html

How to test: Tester enters and saves the data, finds the name in the address book, and clicks on it. One sees a read-only view of the contact-entry screen, with all data previously entered.

The elements in this User Story are:

1. **Name:** The Name is a descriptive phrase or sentence. The example uses a basic "Role-Action-Reason" organization. Another common style, popularized by Mike Cohn, follows the template "As a <type of user>, I want <some goal> so that <some reason>." The choice of template is less important than having a workable standard of some kind.
2. **Description:** This is a high-level (low-detail) description of the need to be met. For functional (user-facing) requirements, the description is put in narrative form. For non-functional requirements, the description can be worded in any form that is easy to understand. In both cases, the key is that the level of detail is modest, because the fine details are worked out during the implementation phase, in discussions between team members, product owners, and anyone else who is involved. (This is one of the core concepts of Scrum: Requirements are specified at a level that allows rough estimation of the work required to implement them, not in detail.)

3. **Screens and External Documents:** If the Story requires user-interface changes (especially non-trivial ones), the Story should contain or link to a prototype of the changes. Any external documents required to implement the Story should also be listed.
4. **How to test:** The implementation of a Story is defined to be complete if, and only if, it passes all acceptance tests developed for it. This section provides a brief description of how the story will be tested. As for the feature itself, the description of testing methods is short, with the details to be worked out during implementation, but we need at least a summary to guide the estimation process.

There are two reasons for including the information about how to test the Story. The obvious reason is to guide development of test cases (acceptance tests) for the Story. The less-obvious, but important, reason, is that the Team will need this information in order to estimate how much work is required to implement the story (since test design and execution is part of the total work).

Story

Not all requirements for new development represent user-facing features, but do represent significant work that must be done. These requirements often, but not always, represent work that must be done to support user-facing features. We call these non-functional requirements "Technical Stories." Technical Stories have the same elements as User Stories, but need not be cast into narrative form if there is no benefit in doing so. Technical Stories are usually written by Team members, and are added to the Product Backlog. The Product Owner must be familiar with these Stories, and understand the dependencies between these and User Stories in order to rank (sequence) all Stories for implementation.

Defect

A Defect, or bug report, is a description of a failure of the product to behave in the expected fashion. Defects are stored in a bug-tracking system, which may or may not be physically the same system used to store the Product Backlog. If not, then someone (usually the Product Owner) must enter each Defect into the Product Backlog, for sequencing and scheduling.

Scrum Roles

The three roles defined in Scrum are the ScrumMaster, the Product Owner, and the Team (which consists of Team members). The people who fulfill these roles work together closely, on a daily basis, to ensure the smooth flow of information and the quick resolution of issues.

ScrumMaster

The ScrumMaster (sometimes written "Scrum Master," although the official term has no space after "Scrum") is the keeper of the process. The ScrumMaster is responsible for making the process

run smoothly, for removing obstacles that impact productivity, and for organizing and facilitating the critical meetings. The ScrumMasters responsibilities include

- Removing the barriers between the development Team and the Product Owner so that the Product Owner directly drives development.
- Teach the Product Owner how to maximize return on investment (ROI), and meet his/her objectives through Scrum.
- Improve the lives of the development Team by facilitating creativity and empowerment.
- Improve the productivity of the development Team in any way possible.
- Improve the engineering practices and tools so that each increment of functionality is potentially shippable.
- Keep information about the Team's progress up to date and visible to all parties.

In practical terms, the ScrumMaster needs to understand Scrum well enough to train and mentor the other roles, and educate and assist other stakeholders who are involved in the process. The ScrumMaster should maintain a constant awareness of the status of the project (its progress to date) relative to the expected progress, investigate and facilitate resolution of any roadblocks that hold back progress, and generally be flexible enough to identify and deal with any issues that arise, in any way that is required. The ScrumMaster must protect the Team from disturbance from other people by acting as the interface between the two. The ScrumMaster does not assign tasks to Team members, as task assignment is a Team responsibility. The ScrumMaster's general approach towards the Team is to encourage and facilitate their decision-making and problem-solving capabilities, so that they can work with increasing efficiency and decreasing need for supervision. The goal is to have a team that is not only empowered to make important decisions, but does so well and routinely.

Product Owner

The Product Owner is the keeper of the requirements. The Product Owner provides the "single source of truth" for the Team regarding requirements and their planned order of implementation. In practice, the Product Owner is the interface between the business, the customers, and their product related needs on one side, and the Team on the other. The Product Owner buffers the Team from feature and bug-fix requests that come from many sources, and is the single point of contact for all questions about product requirements. Product Owner works closely with the team to define the user-facing and technical requirements, to document the requirements as needed, and to determine the order of their implementation. Product Owner maintains the Product Backlog (which is the repository for all of this information), keeping it up to date and at the level of detail and quality the Team requires. The Product Owner also sets the schedule for releasing completed work to customers, and makes the final call as to whether implementations have the features and quality required for release.

Team

The Team is a self-organizing and cross-functional group of people who do the hands-on work of developing and testing the product. Since the Team is responsible for producing the product, it must also have the authority to make decisions about how to perform the work. The Team is therefore self-organizing: Team members decide how to break work into tasks, and how to allocate tasks to individuals, throughout the Sprint. The Team size should be kept in the range from five to nine people, if possible. (A larger number make communication difficult, while a smaller number leads to low productivity and fragility.) Note: A very similar term, "Scrum Team," refers to the Team plus the ScrumMaster and Product Owner.

What is your next step with Agile?

Training and Certifications

We are the largest provider of Agile training in the United States. Our public training courses are available across the country, with locations and dates that are convenient for you. We are Scrum Alliance Registered Education providers and SAFe Gold SPCT partners, and we are confident that we have the right course for you. You can also talk to us about corporate training deals and get your entire team (or company) certified!

Get more resources

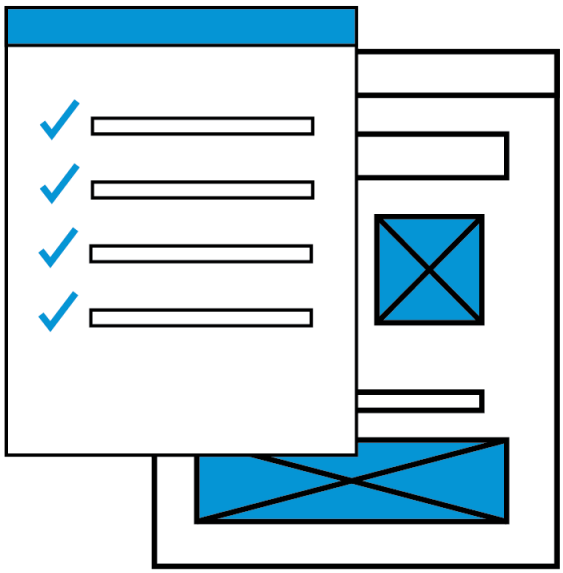
We have plenty more resources to help you along your Agile journey. Visit www.cprime.com to check out our full resource library. We add blogs weekly and add new live and recorded webinars every month!

Get help from our experts

Sometimes you need a little extra help to bring about lasting change. Our experts have deep expertise and a range of specialties. Talk to us about workshops, private training, assessments, or anything else. We are confident we can help. Email learn@cprime.com.

Encyclopedia of Agile

An Indexed glossary of terms used in Agile Software Development



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

The intention of compiling this Encyclopedia of Agile was to create a document that can be used both as an offline reference, and as an online linked reference to some of the top thought leaders in the software industry.

The terms and definitions found in this encyclopedia are public domain. The wording of the definitions were taken from the following sources, with reference links to these sources for more in depth coverage of the terminology definitions and etymology.

While the content is not new, or original, I hope that you find the various indexes and references helpful in your search for understanding of all things Agile.

http://www.wikipedia.org/

Scrum.org

Agile Glossary

ScrumAlliance

Agile Glossary

Agile Glossary

Agile Glossary

Scrum Terminology Index

Scrum Roles

Scrum Activities

Scrum Artifacts

Types Of Stories

Agile Terminology Index

Agile Terminology Index

Agile Terminology Index

1 A

1.1 Acceptance Testing

Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

Reference: Wikipedia

Return To Glossary

1.2 * Adaptive

Return To Glossary

1.3 * Affinity Estimating

Affinity Estimating is a process to quickly estimate a large number of stories with high level SWAG estimates relative to other stories in the same project. To popular tactics are to estimate using either relative or absolute points.

Estimating with absolute units:

When estimating with absolute units the facilitator will quickly review several stories, asking the team for a flash vote on size (1,2,3,5,8,13,Epic). each new story is compared to the previously voted stories for equivalent size. Each vote is a flash vote, no more than 60 seconds discussion. As each story is estimated the story card is dropped onto the specified stack. By the end of the exercise al stories have been assigned and absolute story point size

Estimating relative units:

When estimating with relative units the delivery team works in parallel, each selecting a stack of stories and sorting them on a wall, floor or table in relative size, smallest to largest. As the team members work through their stacks they can reference stories placed by other team members, possibly moving those stories to a new location in the continuum. After all stories have been placed and the team has reviewed the relative sorting order of the entire backlog the continuum is

translated to story points by marking equal gradations along the continuum (1,2,3,5,8,13,Epic). A this point the team can reference the established boundaries and move stories to one side or the other of a boundary line according to their best judgement. By the end of the process all stories will be assigned a relative story point size.

See Also: Estimation, Release Planning

References: SolutionsIQ

Return To Glossary

1.4 * Agile

Return To Glossary

1.5 Agile Development Practices

Procedures and techniques used to conduct Agile software development. Although there is no canonical set of Agile practices, most Agile practitioners adopt some subset of Scrum and XP practices.

Broadly speaking, any practice or technique that facilitates the values and principles set forth in the Agile manifesto can be considered an Agile practice.

The most popular agile methodologies include:

Extreme Programming (XP)

Scrum

Crystal,

Dynamic Systems Development Method (DSDM)

Lean Development

Feature Driven Development (FDD).

All Agile methods share a common vision and core values of the Agile Manifesto.

Some other well-known agile software development methods include:

Agile modeling

Agile Unified Process (AUP)

Essential Unified Process (EssUP)

Open Unified Process (Open UP)

Velocity Tracking

See Also: Agile Manifesto

References:

Return To Glossary

1.6 Agile Estimation

Agile estimation is a process of agreeing on a size measurement for the stories in a product backlog. Agile estimation is done by the team, usually using Planning Poker.

Return To Glossary

1.7 Agile Manifesto

A philosophical foundation for effective software development, the Agile Manifesto was created by representatives from Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others sympathetic to the need for an alternative to documentation-driven, heavyweight software development processes. It reads, in its entirety, as follows:

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.

Twelve principles underlie the Agile Manifesto, including:

Customer satisfaction by rapid delivery of useful software

Welcome changing requirements, even late in development

Working software is delivered frequently (weeks rather than months)

Working software is the principal measure of progress

Sustainable development, able to maintain a constant pace

Close, daily co-operation between business people and developers

Face-to-face conversation is the best form of communication (co-location)

Projects are built around motivated individuals, who should be trusted

Continuous attention to technical excellence and good design

Simplicity

Self-organizing teams

Regular adaptation to changing circumstances

Some of the manifesto's authors formed the Agile Alliance, a non-profit organization that promotes software development according to the manifesto's principles.

References: Wikipedia, AgileManifesto.org, 12 Agile Principles

Return To Glossary

1.8 Agile Methods

See Agile Development Practices

Return To Glossary

1.9 Agile Methodology

Agile Methodology is an umbrella term for several iterative and incremental software development methodologies.

See Agile Development Practices

Return To Glossary

1.10 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.

Return To Glossary

1.11 Agile Planning Basics

The four basics of Agile planning are: Product Backlog, Estimates, Priorities and Velocity.

- Estimates answer the question: "How long will it take or how many can we do by a given date?"
- Priorities answer the question: "Which capabilities are most important?
- The Product Backlog answers the question: "What capabilities are needs for financial success?"
- Velocity answers the question: "How much can the team complete in a Sprint?"

Return To Glossary

1.12 Agile Project Management

The style of project management used to support Agile software development. Scrum is the most widely used Agile project management practice. XP practices also include practices that support Agile project management. Essential feature of Agile project management include:

i. Iterative development cycles
ii. Self-organizing teams
iii. Multi-level planning
iv. Dynamic scope
v. Frequent collaboration with customer and/or business sponsors

Related links: Wikipedia

Return To Glossary

1.13 Agile Software Development

Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self organizing team, cross functional teams.

The development of software using Agile development practices and Agile project management.

Features of Agile software development include a heavy emphasis on collaboration, responsiveness to change, and the reduction of waste throughout the development cycle.

Agile software development (ASD) focuses on keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready.

References: Wikipedia

Return To Glossary

1.14 * Agile Unified Process

Return To Glossary

1.15 * Agilista

Return To Glossary

1.16 * Agility

Return To Glossary

1.17 Alignment

Organizations with production dependencies across department boundaries run the risk of falling out of phase (or alignment). Alignment includes any actions or policies that exist so that a process or activity in one section of the organization is congruent with the organization's or business unit's governing mission. The lack of business/IT alignment is a chronic problem for many organizations and frequently the root cause of systemic software delivery failure. Agile development practices are designed to address many of the root causes of misalignment between IT and the business.

References: Wikipedia

Return To Glossary

1.18 ALM

See: Application Lifecycle Management

Return To Glossary

1.19 * Anchoring

Return To Glossary

1.20 Application Lifecycle Management

"Application Lifecycle Management (ALM) is a continuous process of managing the life of an application through governance, development and maintenance." (Wikipedia)

When Agile software development is introduced into an organization it generally requires substantial changes in the organization's ALM tools and policies, which are typically designed to support alternative methodologies such as Waterfall.

References: Wikipedia

Return To Glossary

2 B

2.1 Backlog

The generic term for a repository of requirements (stories / work items) that define a system and it's many parts. The outermost scope of work defined is the Product Backlog, which defines all requirements/features/defects/stories for a given product. A Product Backlog is subdivided into one or more Release Backlogs. During Sprint planning the delivery team estimates the top most Backlog Items in the current Release Backlog and assigns them to their Sprint Backlog where they are tracked and implemented for the current sprint.

See also: Product Backlog Item, Task, Iteration, Sprint, Sprint Backlog Product Owner, Planning Game

References: Wikipedia

Return To Glossary

2.2 Backlog Item

See: Product Backlog Item

References: Wikipedia

Return To Glossary

2.3 Backlog Item Effort

Some Scrum practitioners estimate the effort of product backlog items in ideal engineering days, but others prefer less concrete backlog effort estimation units. Alternative units might include story points, function points, or "t-shirt sizes" (1 for small, 2 for medium, etc). The advantage of more vague units is that they're explicit about the distinction that product backlog item effort estimates are estimates of effort, not duration. Also, estimates at this level are rough guesses that should never be confused with actual working hours (Note that sprint tasks are distinct from product backlog items and task effort remaining is always estimated in hours).

References: SolutionsIQ:Backlog Item Effort

Return To Glossary

2.4 Backlog Grooming

Backlog grooming is both an ongoing process and the name for a meeting:

The process of adding new user stories to the backlog, re-prioritizing existing stories as needed, creating estimates, and deconstructing larger stories into smaller stories or tasks.

A meeting or ceremony that occurs regularly within a team's iteration cycle. Scrum Alliance founder Ken Schwaber recommends that teams allocate 5% of their time to revisiting and tending to the backlog. Backlog grooming is the term favored by the Scrum Alliance, although Scrum co-founder Jeff McKenna and Australian CST Kane Mar prefer to call this ceremony Story Time.

Return To Glossary

2.5 Big Ball of Mud

"A Big Ball of Mud is a haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated. The overall structure of the

system may never have been well defined. If it was, it may have eroded beyond recognition. Programmers with a shred of architectural sensibility shun these quagmires. Only those who are unconcerned about architecture, and, perhaps, are comfortable with the inertia of the day-to-day chore of patching the holes in these failing dikes, are content to work on such systems."

- Brian Foote and Joseph Yoder, Big Ball of Mud.

References: Wikipedia, Big Ball of Mud

Return To Glossary

2.6 Big Visible Charts

Big visible charts are exactly what you would think they would be: Big charts posted near the agile team that describe in different ways the team's progress. Big visible charts not only can be useful tools for the team but also make it easier for any stakeholder to learn how the team is progressing. Big visible charts are an important tool for implementing the essential agile values of transparency and communication.

References: XPProgramming.com

Return To Glossary

2.7 * Blocked

See Also:

Return To Glossary

2.8 Bottleneck

Any resource or process whose capacity is less than or equal to the demand placed on it, thus constraining the flow of work or information through the process.

See Also: Kanban

References: Wikipedia

Return To Glossary

2.9 Branching

"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."

References: Accurev.com

Return To Glossary

2.10 Breaking the Build

When a developer adds changes to the source code repository that result in the failure of a subsequent build process, the developer has "broken the build." Avoiding breaking the build is a commitment generally required by agile software developers and integral to the XP practice continuous integration.

The build is broken if the build process cannot successfully completed for any number of reasons including (but not limited to) failure to compile, compiling with unacceptable warnings, or the failure of any number of (usually) automated software tests. The more comprehensive the build process, the higher the threshold for breaking the build.

If a code submission does result in breaking the build, the developer should immediately remove the cause. If the build breaks but the immediate cause is not self-evident, a frequent practice of established agile development teams is to take immediate action to fix the build.

Return To Glossary

2.11 Build Process

"The amount of variability in implementation makes it difficult to come up with a tight definition of a Build Process, but we would say that a Build Process takes source code and other configuration data as input and produces artifacts (sometimes called derived objects) as output. The exact number and definition of steps depends greatly on the types of inputs (Java versus C/C++ versus Perl/ython/Ruby source code) and the type of desire output (CD image, downloadable zip file or self-extracting binary, etc). When the source code includes a compiled language then the Build Process would certainly include a compilation and perhaps a linking step." (Anthillpro)

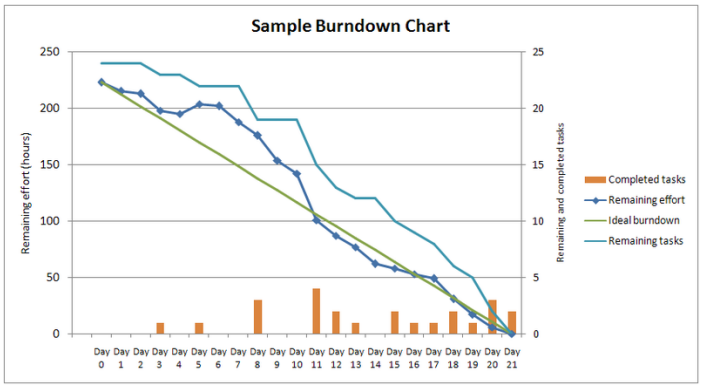
Return To Glossary

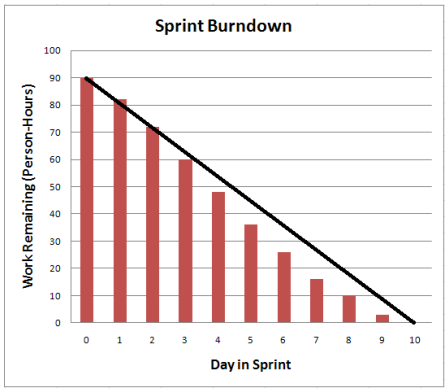
2.12 Burn-Down Chart

A Burn down chart is a chart showing how much work remaining in a sprint. Calculated in hours remaining and maintained by the Scrum Master daily.

A publicly displayed chart that depicts the total task hours remaining per day. It shows where the team stands regarding completing the tasks that comprise the backlog items that achieve the goals of the

sprint. The X-axis represents days in the sprint, while the Y-axis is effort remaining (usually in ideal engineering hours). To motivate the team, the sprint burn-down chart should be displayed prominently. It also acts as an effective information radiator. A manual alternative to this is a physical task board. Ideally, the chart burns down to zero by the end of the sprint. If the team members are reporting their remaining task hours realistically, the line should bump up and down.





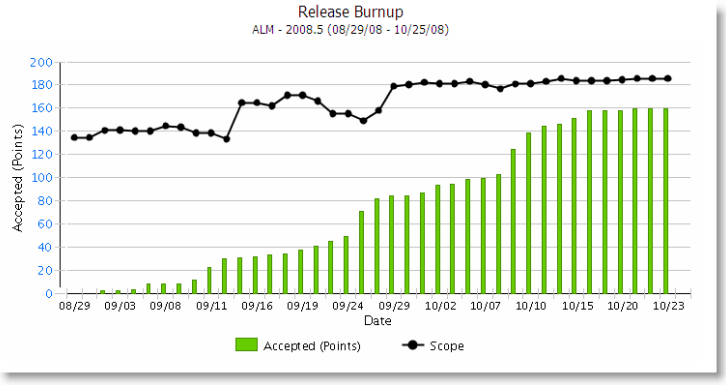
See Also: Burn-Up Chart

References: Wikipedia

Return To Glossary

2.13 Burn-Up Chart

Representation of the amount of stories completed, with points plotted on an X and Y axis that map an upward trend of work completed until reaching 100%.



Return To Glossary

2.14 Business Alignment

See: Alignment

Return To Glossary

2.15 Business Value

Each user story in the Product Backlog should have a corresponding business value assigned. Typically assign (L,M,H) Low, Medium, High. Product Owner prioritizes Backlog items by highest value.

An informal term that includes all forms of value that determine the health and well-being of the firm in the long run. It expands the concept of value of the firm beyond economic value to include other forms of value such as employee value, customer value, supplier value, channel partner value, alliance partner value, managerial value, and societal value. In the context of agile development, it is what management is willing to pay for and a way to identify the value of "work" or a story.

References: Wikipedia

Return To Glossary

3 C

3.1 Capacity

Capacity is the Number of Teammates (Productive Hours x Sprint Days).

Example:

- Team size is 4,
- Productive hours per person per day are 5,
- Sprint length is 30 days.
- Capacity = 4(5x30) = 600 hours.

Return To Glossary

* CANI

Constant And Never-ending Improvement

See Also: Kaizen, Inspect & Adapt, Retrospective

Return To Glossary

3.2 * Card-Conversation-Confirmation

XP Practices for generating a well groomed backlog, elaborating story contents and validating completed results.

"User stories have three critical aspects. We can call these Card, Conversation, and Confirmation."

Ron Jeffries, 2001

References: Card-Conversation-Confirmation

Return To Glossary

3.3 Certified ScrumMaster

Someone who is acting in the role of ScrumMaster on a Scrum team and who has attended a two-day Certified ScrumMaster (CSM) class to obtain certification.

References: Wikipedia

Return To Glossary

3.4 Chicken

Scrum slang for someone who is interested in a project but has no responsibility for working on a task in the active iteration. They may observe team meetings but cannot vote or talk.

Chickens are the people that are not committed to the project and are not accountable for deliverables.

References: Wikipedia

See also: Pig

Return To Glossary

3.5 Code Smell

"Any symptom in the source code of a computer program that indicates something may by wrong." (Wikipedia)

Common code smells are often used to diagnose the quality of legacy code. Code smells generally indicate that the code should be refactored or the overall design should be reexamined.

See Also: Refactoring

References: Wikipedia

Return To Glossary

3.6 Colocation

Refers to development teams located and working in the same location. When possible colocation is desirable since it facilitates face-to-face collaboration, an important features of Agile software development. Contrast with distributed development team.

See Also: Colocation, Agile software Development, Distributed Development Team

References: Wikipedia

Return To Glossary

3.7 Continuous Integration

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly." (MartinFowler.com)

References: Wikipedia

Return To Glossary

3.8 Cross-Functional Team

Team comprised of members with all functional skills and specialties necessary to complete a project from start to finish.

References: Wikipedia

Return To Glossary

3.9 * Crystal

1990s

Return To Glossary

3.10 Customer

The recipient of the output (product, service, information) of a process. Customers may be internal or external to the organization. The customer may be one person, a department, or a large group. Internal customers (outside of Information Technology) are sometimes called the "Business."

References: Wikipedia

Return To Glossary

4 D

4.1 Daily Scrum

See: Standup Meeting

Return To Glossary

4.2 Daily Standup

See: Standup Meeting

Return To Glossary

4.3 Defect

A defect is a failure or bug of the product to behave in the expected fashion. Defects are stored in a bug-tracking system, which may or may not be physically the same system used to store the Product Backlog. If not, then someone (usually the Product Owner) must enter each Defect into the Product Backlog, for sequencing and scheduling.

See Also: Story, User Story, Technical Story, Spike, Tracer Bullet

Return To Glossary

4.4 Definition of Done

The criteria for accepting work as completed. Specifying these criteria is the responsibility of the entire team, including the business. Generally, there are three levels of "Done" (also known as Done-Done-Done):

Done: Developed, runs on developer's box

Done: Verified by running unit tests, code review, etc.

Done: Validated as being of deliverable quality with functional tests, reviews, etc.

However, the exact criteria for what constitutes "Done" varies to meet the specific needs of different organizations and initiatives. An important agile principle is to deliver (potentially) releasable software after every iteration. The definition of done is a key component of Agile project governance used to help teams comply with this principle.

Return To Glossary

4.5 Delivery Team

In agile software development, the delivery team refers to the cross-functional group of people that have made a collective commitment to work together to produce the work product and improve their performance over time. In addition to software development and test roles, the team may include any skill set necessary to deliver the work product.

The delivery team usually includes people skilled to understand customer requirements and conduct software design, coding and testing. Additional skills (e.g. UI design, usability, etc.) may also be included, especially when they are integral to the software release.

The delivery team is encouraged to be self-organizing and to take collective responsibility for all work commitments and outcomes. Delivery teams respond to requirements (often presented as user stories) by collectively defining their tasks, task assignments, and level of effort estimates.

The ideal size for a delivery team adheres to the magic number seven plus or minus two rule.

See Also: Scrum Team, Product Owner, ScrumMaster

References: Wikipedia

Return To Glossary

4.6 * Dependency Injection Principle

See Also: SOLID OOD Principles:

Single Responsibility Principle

Open Closed Principle,

Liskov Substitution Principle,

Interface Segregation Principle,

Dependency Injection Principle

Return To Glossary

4.7 Design Pattern

"A design pattern is a general reusable solution to a commonly occurring problem in software design." (Wikipedia)

Return To Glossary

4.8 Distributed Development Team

Refers to development teams that work on the same project but are located across multiple geographic locations or work sites. Distributed development teams are becoming the norm for today's software projects. When co-location is not an option, distributed teams are faced with the challenge of keeping software projects on track and keeping remote developers engaged collaboratively. Agile development is more difficult for distributed teams and generally require that special practices are adopted that mitigate the inherent risks of distributed development.

See Also: Colocation

References: Wikipedia

Return To Glossary

4.9 Distributed Scrum

See Distributed Development Team

Return To Glossary

4.10 Domain Model

Information model describing the application domain that creates a shared language between business and IT

References: Wikipedia

Return To Glossary

4.11 DSDM

See Dynamic Systems Development Method

Return To Glossary

4.12 * Dynamic Systems Development Method

Return To Glossary

5 E

5.1 * Earned Value Chart

Return To Glossary

5.2 Emergence

Emergence is an attribute of complex systems. When applied to software development, it is the principle that the best designs and the best ways of working come about over time through doing the work, rather than being defined in advance as part of an over-arching specification or detailed project plan.

See Also: Self-Organization

References: Wikipedia

Return To Glossary

5.3 Empiricism

Empiricism is the principle that knowledge is acquired through our experience, which we obtain through our senses. Empiricism is the cornerstone of all scientific inquiry and the approach used by Agile teams to identify emergent requirements and incrementally develop software.

See Also: Inspect and Adapt

References: Wikipedia

Return To Glossary

5.4 Epic

A very large user story that is eventually broken down into smaller stories. Epics are often used as placeholders for new ideas that have not been thought out fully or whose full elaboration has been deferred until actually needed. Epic stories help agile development teams effectively manage and groom their product backlog.

See Also: Story, Backlog, Backlog Grooming

References: SolutionsIQ: Epic

Return To Glossary

5.5 EssUP

See Essential Unified Process

Return To Glossary

5.6 * Essential Unified Process:

Return To Glossary

5.7 Estimation

The process of agreeing on a size measurement for the stories or tasks in a product backlog. On agile projects, estimation is done by the team responsible for delivering the work, usually using a planning game.

Estimates on stories are made in abstract story points.

Estimates on tasks are made in hours.

See Also: Story, Backlog, Planning Game, Tasks, Story Points

References: Wikipedia

Return To Glossary

5.8 * Estimate to Complete Chart

Return To Glossary

5.9 Extreme Programming

1996

A software development methodology adhering to a very iterative and incremental approach, Extreme Programming 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 (time-boxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

XP consists of a number of integrated practices for developers and management - the original twelve practices of XP include:

1. Small Releases
2. On-site Customer
3. Sustainable Pace
4. Simple Design
5. Continuous Integration
6. Unit Testing
7. Coding Conventions
8. Refactoring Mercilessly
9. Test-Driven Development
10. System Metaphor
11. Collective Code Ownership
12. Pair Programming

Most successful Agile practitioners adopt some subset of XP practices, often in conjunction with Scrum.

See Also: Unit Testing, Refactoring, Extreme Programming (XP), Time-box

References: Wikipedia

Return To Glossary

6 F

6.1 Fail-Fast

"A property of a system or module with respect to its response to failures. A fail-fast system is designed to immediately report at its interface any failure or condition that is likely to lead to failure." (Wikipedia)

Return To Glossary

6.2 Feature

A coherent business function or attribute of a software product or system. Features are large and chunky and usually comprise many detailed (unit) requirements. A single feature typically is implemented through many stories. Features may be functional or non-functional; they provide the basis for organizing stories.

See also: Minimum Marketable Features, User Story

References: Wikipedia

Return To Glossary

6.3 FDD

See Feature Driven Development

Return To Glossary

6.4 * Feature Driven Development

Return To Glossary

6.5 Fibonacci Sequence

A sequence of numbers in which the next number is derived by adding together the previous two (e.g. 1, 2, 3, 5, 8, 13, 21, 34...). The sequence is used to size stories in Agile estimation techniques such as Planning Poker.

References: Fibonacci Sequence

Return To Glossary

6.6 Flow

Continuous delivery of value to customers (vs. big-batch, big-release, big-bang).

See also: Planning Poker, Fibonacci Sequence

References: Wikipedia

Return To Glossary

6.7 * Forked Development

Return To Glossary

6.8 Fog Of War

"The fog of war is a term used to describe the uncertainty in situation awareness experienced by participants in military operations. The term seeks to capture the uncertainty regarding own capability, adversary capability, and adversary intent during an engagement, operation, or campaign."

Wikipedia

In Agile the fog of war refers to the increasing uncertainty of estimates.

As stories increase in size the level of confidence in the estimates decreases.

As stories are scheduled farther away for implementation the level of confidence in those estimates decreases significantly. For this reason it is not reasonable to depend on estimates for stories expected to be implemented more than a month out. As those stories come into view on the near term schedule new estimates can be made with greater confidence.

Generally the best estimates are given during sprint planning sessions where the story is expected to be implemented that sprint.

References: Wikipedia

Return To Glossary

6.9 * Functional Test

See Also:

Return To Glossary

7 G

8 H

8.1 * Ha

See Also: Shu-Ha-Ri, Shu, Ri

Return To Glossary

9 I

9.1 Impediment

In Scrum: Anything that prevents a team member from performing work as efficiently as possible is an impediment. Each team member has an opportunity to announce impediments during the daily standup meeting. The ScrumMaster is charged with ensuring impediments are removed. ScrumMasters often arrange sidebar meetings, Parking Lot, when impediments cannot be resolved on the spot in the daily Scrum meeting.

See also: Scrum, Daily Standup, ScrumMaster, Parking Lot

References: Wikipedia

Return To Glossary

9.2 INVEST

Criteria for well written user stories. Every user story should satisfy the following INVEST principles:

a. Independent
b. Negotiable
c. Valuable
d. Estimable
e. Small
f. Testable.

Return To Glossary

9.3 * Integration Test

See Also:

Return To Glossary

9.4 Inspect and Adapt

"Inspect and Adapt" is a slogan used by the Scrum community to capture the idea of discovering over the course of a project emergent software requirements and ways to improve the overall performance of the team. It neatly captures the both the concept of empirical knowledge acquisition and feedback-loop-driven learning.

See Also: CANI, Kaizen, Retrospective, Empiricism

Return To Glossary

9.5 * Interface Segregation Principle

See Also: SOLID OOD Principles:

Single Responsibility Principle,

Open Closed Principle,

Liskov Substitution Principle,

Interface Segregation Principle

Dependency Injection Principle

Return To Glossary

9.6 * Iron Triangle

Return To Glossary

9.7 IT Alignment

See: Alignment

Return To Glossary

9.8 Iteration

A period (from 1 week to 2 months in duration) during which the Agile development team produces an increment of completed software. All system lifecycle phases (requirements, design, code, and test) must

be completed during the iteration and then (empirically) demonstrated for the iteration to be accepted as successfully completed. At the beginning of the iteration, the business or the product owner identifies the next (highest priority) chunk of work for the team to complete. The development team then estimates the level of effort and commits to completing a segment of work during the iteration. During the iteration, the team is not expected to change objectives or respond to change requests. However, at the front end of the next iteration the business or product owner is free to identify any new segment of work as the current highest priority.

See also: Sprint, Definition of Done, Velocity, Task Board, Kanban

References: Wikipedia

Return To Glossary

10 J

11 K

11.1 Kanban

Kanban is a tool derived from lean manufacturing and is associated with the branch of agile practices loosely referred to as Lean software development. Like a task board, Kanban visually represents the state of work in process. Unlike a task board, the Kanban constrains how much work in process is permitted to occur at the same time. The purpose of limiting work in process is to reduce bottlenecks and increase throughput by optimizing that segment of the value stream that is the subject of the Kanban. Task boards simply illustrate work in process without necessarily deliberately how much of work in process may occur at any given time, although the same effect may be achieved through the organic self-organization of the team.

A principle difference between Kanban and Scrum is that Scrum limits work in process through time-boxing (i.e. the sprint) and Kanban limits work in process by limiting how much work may occur at one time (e.g. N tasks or N stories).

References: Wikipedia

Return To Glossary

11.2 * Kata

See Also:

Return To Glossary

11.3 * Kaizen

See Also: CANI, Inspect & Adapt, Retrospective

Return To Glossary

12 L

12.1 Lean Software Development

An adaption of Lean manufacturing principles and practices to the software development domain. Lean software development (also known as Lean-Agile) is focused on reducing (lean) waste and optimizing the software production value stream. In large part, the principles and practices of lean software development are congruent with other well-known Agile practices such as Scrum and extreme programming. However, in some cases they use different means to obtain the same end. For example, Scrum and Kanban (a lean technique) both reduce work in process (a lean waste) but use different techniques to accomplish this objective.

Authors Mary and Tom Poppendieck bring Lean Manufacturing Principles to Software Development.

References: Wikipedia

Return To Glossary

* Levels of Planning, 5

In Scrum there are 5 levels of planning identified as :

1. Vision
2. Roadmap
3. Release
4. Sprint
5. Daily

Return To Glossary

12.2 * Liskov Substitution Principle

See Also: SOLID OOD Principles: SingleResponsibilityPrinciple, OpenClosedPrinciple, InterfaceSegregationPrinciple, DependencyInjectionPrinciple

Return To Glossary

12.3 * Load Test

See Also:

Return To Glossary

13 M

13.1 Minimum Marketable Features

The smallest set of functionality that must be realized in order for the customer to perceive value. A "MMF" is characterized by the three attributes: minimum, marketable, and feature. A feature is something that is perceived, of itself, as value by the user. "Marketable" means that it provides significant value to the customer; value may include revenue generation, cost savings, competitive differentiation, brand-name projection, or enhanced customer loyalty. A release is a collection of MMFs that can be delivered together within the time frame.

References: Wikipedia

Return To Glossary

14 N

15 O

15.1 OpenUP

See Open Unified Process

Return To Glossary

15.2 * Open Unified Process

Return To Glossary

15.3 * Open Closed Principle

See Also: SOLID OOD Principles: SingleResponsibilityPrinciple, LiskovSubstitutionPrinciple, InterfaceSegregationPrinciple, DependencyInjectionPrinciple

Return To Glossary

15.4 * Osmotic Communication

Return To Glossary

16 P

16.1 Pair Programming

"An Agile software development technique in which two programmers work together at one workstation. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. and the person reviewing the code is called the observer or navigator. The two programmers switch roles frequently." (Wikipedia)

Pair programming is one of the original 12 extreme programming practices. As counter-intuitive as it may seem to the uninitiated, pair programming is more productive than two individuals working independently on separate tasks.

Return To Glossary

16.2 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.

Return To Glossary

16.3 * Pareto Principle

References: BetterExplained, Wikipedia

Return To Glossary

16.4 * Parking Lot

Return To Glossary

16.5 Pattern

See: Design Pattern

Return To Glossary

* Performance Test

See Also: Profiling, Acceptance Test, Functional Test, SystemTest, Integration Test, Unit Test, Stress Test, Load Test

Return To Glossary

16.6 Pig

Scrum slang. Someone who is responsible for doing a task on an active iteration. It comes from the joke, "A chicken and a pig talk about breakfast. The chicken says, 'Let's have bacon and eggs.' The pig replies, 'That's fine for you. You are just making a contribution, but I have to be fully committed.'" Pigs are actively involved in the project.

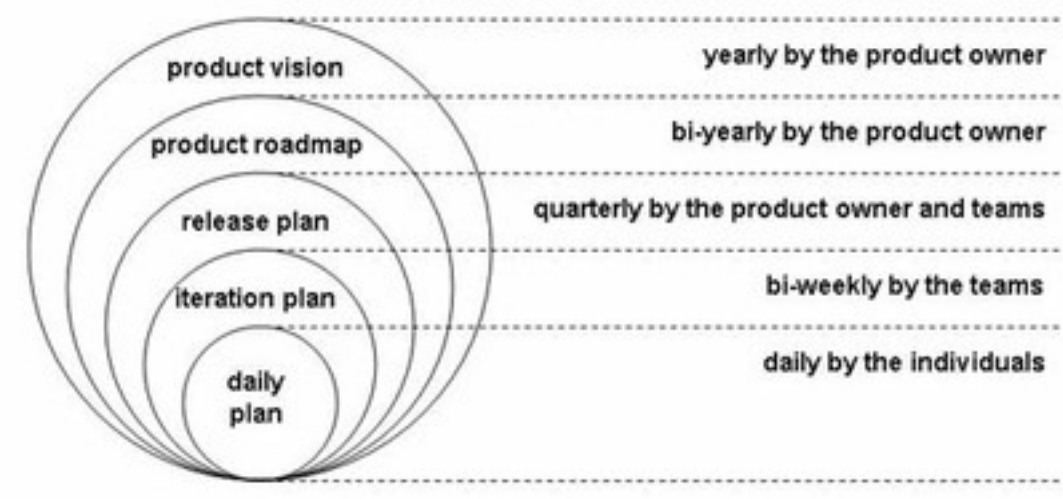
See also: Chicken

References: Wikipedia

Return To Glossary

16.7 Planning

5 Levels of Planning



plans are useless but planning is indispensable. Dwight Eisenhower

See Also: Product Vision, Product Roadmap, Release Plan, Sprint Plan, Daily Standup

References: AgileJournal, RallyDev

Return To Glossary

16.8 Planning Game

"The main planning process within extreme programming is called the Planning Game. The game is a meeting that occurs once per iteration, typically once a week. The planning process is divided into two parts." (Wikipedia)

In XP, the planning game includes iteration (or sprint) planning and release planning. In scrum, sprint and release planning are two of the five levels of planning used in Agile projects.

See also: Sprint Planning Meeting, Release Planning

References: Wikipedia

Return To Glossary

16.9 Planning Poker

"Planning Poker is a consensus-based technique for estimating, mostly used to estimate effort or relative size of tasks in software development." (Wikipedia)

Planning poker is a game used to apply estimates to stories. It uses a voting approach designed to avoid influence bias (anchoring).

How it Works:

1. Each estimator selects a set of cards.
2. Facilitator reads item to be estimated, and moderates a brief discussion to clarify details.
3. Facilitator calls for estimates. Each estimator places estimate face down, hiding the value.
4. Facilitator calls for vote, and all estimators turn over cards at the same time.
5. If all cards agree, their value is recorded as the estimate.
6. Otherwise, facilitator asks high and low estimators to explain their reasoning, and moderates a brief discussion to clarify issues.
7. Repeat 3-6 until estimates converge

References: Wikipedia

Return To Glossary

16.10 * Pragmatic Programming

See Also: Agile Development Methods

Return To Glossary

16.11 * Predictive

See Also: Adaptive, Reactive

Return To Glossary

16.12 * Prioritization

See Also: Product Backlog

Return To Glossary

16.13 * Process Framework

Return To Glossary

16.14 Product

Broadly speaking, product refers to a collection of tangible and intangible features that are integrated and packaged into software releases that offer value to a customer or to a market. The term "product" is often used in Agile software development to denote the software that is the subject of the iteration or release. As such, "product" is generally used interchangeably with other names for software release including "software release", "system", or "business application."

References: Wikipedia

Return To Glossary

16.15 Product Backlog

The product backlog (or "backlog") is the requirements for a system, expressed as a prioritized list of product backlog Items. These included both functional and non-functional customer requirements, as well as technical team-generated requirements. While there are multiple inputs to the product backlog, it is the sole responsibility of the product owner to prioritize the product backlog.

During a Sprint planning meeting, backlog items are moved from the product backlog into a sprint, based on the product owner's priorities.

See: Backlog

Return To Glossary

16.16 * Product Backlog Item

A unit of work, usually a story or a task, listed on the project backlog.

See Also: Product Backlog, Backlog, Backlog Item, Story, Task

Return To Glossary

16.17 Product Owner

Product Owner is one of the key roles in Scrum. The product owner is the primary business representative who represents the business stakeholders' "voice of the customer" and the "voice of the business" to the sprint team. The responsibilities of the Product Owner include:

i. Establishing, nurturing, and communicating the product vision
ii. Creating and leading a team of developers to best provide value to the customer
iii. Monitoring the project against its ROI goals and an investment vision
iv. Making decisions about when to create an official release

The product owner is a role rather than a position. Consequently, several people likely participate in the product owner role for larger projects.

References: Wikipedia

Return To Glossary

16.18 * Product Roadmap

Return To Glossary

16.19 Product Vision

The product vision is one of the five levels of planning.

A product vision is a brief statement of the desired future state that would be achieved through the project initiative. The product vision may be expressed in any number of ways including financial performance, customer satisfaction, market share, functional capability, etc. The product vision is typically the responsibility of executive sponsorship and is articulated to the Agile development team by the business and by the product owner, if the team is using Scrum.

References: Wikipedia:Elevator Pitch

See also: Product

Return To Glossary

16.20 * Productivity

Return To Glossary

16.21 * Profiling

See Also: Performance Test

Return To Glossary

17 Q

18 R

18.1 * Reactive

Return To Glossary

18.2 Refactoring

Changing existing software code in order to improve the overall design. Refactoring normally doesn't change the observable behavior of the software; it improves its internal structure. For example, if a programmer wants to add new functionality to a program, she may decide to refactor the program first to simplify the addition of new functionality in order to reduce technical debt.

Refactoring is one of the original twelve extreme programming practices and is considered critical for incrementally maintaining technical quality on Agile development projects.

See Also: Code Smell, Extreme Programming, Technical Debt, Design Pattern

References: Wikipedia

Return To Glossary

18.3 Release (Software)

The movement of a software product or system from development into production. One principle of Agile development is to focus on releasing software into productive use as soon as a minimum marketable feature set can be delivered, and then proceeding with frequent incremental releases. This is in contrast to alternative project approaches where most requirements are delivered in one "big bang" release.

It is desirable in Agile development to produce releasable software after every iteration (or sprint), even if the code is not actually put into production for use by end-users.

See Also: Minimum Marketable Features

Return To Glossary

18.4 Release Backlog

18.4.1.1.1 TBD

See Also:

Return To Glossary

18.5 Release Management

18.5.1.1.1 TBD

See Also:

Return To Glossary

18.6 Release Plan

The release plan is a schedule for releasing software into productive use. Typical release plans include the key features to be delivered, along with corresponding release dates. Release plans may also expose key milestones or dependencies that parallel project activities. In agile development, release plans can be mapped back to the iterations (sprints) that implement the released features.

See Also: Release, Release Planning

References: Wikipedia

Return To Glossary

18.7 Release Planning

Release planning refers to planning activities used to estimate when software will be released into product use. Activities include projecting the level of effort in terms of the number of iterations that will be necessary to deliver the desired features. This is typically done by extrapolating the development team's performance on the basis of its velocity.

A release planning meeting that brings together all parties that have a stake in the outcome and have some kind of delivery responsibility to achieve the release is often necessary to produce a viable release plan. This is especially the case when several development and non-development production efforts are running in parallel with possible dependencies.

Release planning is one of the five levels of planning.

See Also: Release Plan, Release, Sprint, Velocity

References: Wikipedia

Return To Glossary

18.8 Research Story

18.8.1.1.1 TBD

Return To Glossary

18.9 Resources

18.9.1.1.1 TBD

Return To Glossary

18.10 Retrospective

A time-boxed meeting held at the end of an iteration, or at the end of a release, in which the team examines its processes to determine what succeeded and what could be improved. The retrospective is key to an Agile team's ability to "inspect and adapt" in the pursuit of "continuous improvement." The Agile retrospective differs from other methodologies' "Lessons Learned" exercises, in that the goal is not to generate a comprehensive list of what went wrong. A positive outcome for a retrospective is to identify one or two high-priority action items the team wants to work on in the next iteration or release. The emphasis is on actionable items, not comprehensive analysis. Retrospectives may take many forms, but

there is usually a facilitator, who may or may not be a member of the team, and the process is typically broken down into three phases: data gathering, data analysis, and action items.

See Also: Sprint, Release, Inspect & Adapt, Effective Retrospectives, CANI, Kaizen, Inspect & Adapt

References: Wikipedia

Return To Glossary

18.11 * Ri

See Also: Shu-Ha-Ri, Shu, Ha

Return To Glossary

18.12 * ROI

Return To Glossary

18.13 * Ron Dori

See Also:

Return To Glossary

19 S

19.1 * Schedule

Return To Glossary

19.2 * Scope

Return To Glossary

19.3 Scrum

A lightweight process framework originally developed in 1995 by Ken Schwaber and Jeff Sutherland.

Scrum is a framework for the iterative development of complex products, particularly software. Scrum is the most widely recognized Agile framework, and is compatible with other Agile practices like Extreme Programming. Scrum is comprised of a series of short iterations - called sprints - each of which ends with the delivery of an increment of working software. The framework is comprised of:

v. Three roles of the Scrum Team
    1. Product Owner
    2. ScrumMaster
    3. Delivery Team

- Five Time-boxes:
    1. Sprint
    2. Sprint Planning Meeting
    3. Daily Standup Meeting
    4. Sprint Review
    5. Retrospective

- Three artifacts:
    1. Burn-down charts
    2. Product backlog
    3. Sprint backlog

Sometimes the term Scrum is used interchangeably with the term Agile, but this is incorrect. Agile is not a framework, but a broader set of values and principles, while Scrum is a specific framework that fits comfortably under the Agile umbrella.

See Also:

References: ScrumAlliance, Scrum.org, ScrumGuide

Return To Glossary

19.4 ScrumBut

ScrumButs are reasons why teams can't take full advantage of Scrum to solve their problems and realize the full benefits of product development using Scrum. Every Scrum role, rule, and timebox is designed to provide the desired benefits and address predictable recurring problems. ScrumButs mean that Scrum has exposed a dysfunction that is contributing to the problem, but is too hard to fix. A ScrumBut retains the problem while modifying Scrum to make it invisible so that the dysfunction is no longer a thorn in the side of the team.

A ScrumBut has a particular syntax: (ScrumBut)(Reason)(Workaround)

ScrumBut Examples:

"(We use Scrum, but) (having a Daily Scrum every day is too much overead,) (so we only have one per week.)"

"(We use Scrum, but) (Retrospectives are a waste of time,) (so we don't do them.)"

"(We use Scrum, but) (we can't build a piece of functionality in a month,) (so our Sprints are 6 weeks long.)"

"(We use Scrum, but) (sometimes our managers give us special tasks,) (so we don't always have time to meet our definition of done.)"

Sometimes organizations make short term changes to Scrum to give them time to correct deficiencies. For example, "done" may not initially include regression and performance testing because it will take several months to develop automated testing. For these months, transparency is compromised, but restored as quickly as possible.

See Also: Scrum, ScrummerFall, ScrumPlus

References:

Return To Glossary

19.5 ScrummerFall

Waterfall management style using iterations and scrum elements. Waterfall/SDLC have distinct stages (Analysis, Design, Develop, Test, Deploy, Maintenance). Scrum combines all stages in a single sprint (iteration).

ScrummerFall happens when a group attempts to bridge these two concepts: Design/Requirement docs are generated in detail ahead of time. Development is completed each sprint then passed to another team for testing outside of that sprint.

a.k.a: Mini-Waterfall

See Also: Scrum, ScrumBut, ScrumPlus

References:

Return To Glossary

19.6 ScrumPlus

Enhancing the Scrum kernel with additional Agile practices that improve transparency, engineering focus, quality management, release management…

See Also: Scrum, ScrumBut, ScrummerFall,

References:

Return To Glossary

19.7 Scrum Team

The scrum team consists of three roles;

1. ScrumMaster

   Maintains the processes (typically in lieu of a project manager)

2. Product Owner

   Represents the stakeholders and the business

3. Delivery Team

   A cross-functional group who do the actual analysis, design, implementation, testing, etc.

References: Wikipedia:Project Manager

Return To Glossary

19.8 ScrumMaster

The ScrumMaster is responsible for maintaining the Scrum process and the overall health of the team. The ScrumMaster assures that the team is fully functional and productive. The ScrumMaster performs this role by administering the Scrum time-boxes, facilitating the organic self-organization of the team, and removing any obstacles that may be impeding the team's progress.

What the ScrumMaster is not:

The ScrumMaster is not the task master, since the team is responsible for assigning its own tasks.

The ScrumMaster is not the supervisor of the team, since the supervisor/subordinate relationship may impede the organic self-organization of the team.

A good ScrumMaster proactively anticipates problems, opportunities for improvement, and conducts pre-planning so the team can focus on delivering its sprint commitments. The ScrumMaster also keeps the

team honest regarding its commitments and helps the team identify opportunities to improve collaboration.

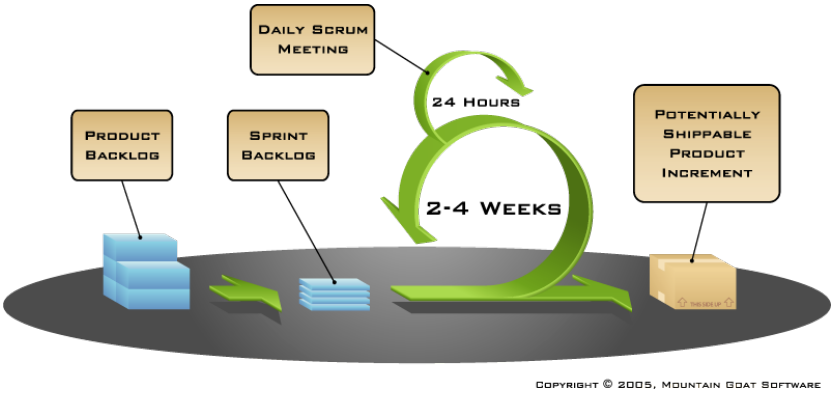
In Scrum, when the Scrum roles are properly fulfilled there is no need for a traditional project manager to supervise the team. Nevertheless, many organizations choose to retain project managers, after they adopt Scrum, to perform functions that extend beyond the scope of the Scrum team functions.

References: Wikipedia

Return To Glossary

19.9 Scrum Snowman

The scrum process of frequent and early feedback cycles is often referred to as the "snowman model" as the cyclic graph resembles a snowman.



References: MountainGoatSoftware

Return To Glossary

19.10 Self-Organization

Self-organization is a property of complex adaptive systems, whereby the organization of the system emerges over time as a response to its environment. In Agile development, particularly in Scrum, self-organization is a property of the agile development team, which organizes itself over time, rather than being ordered by an external force such as a project or development manager. Self-organization also reflects the management philosophy whereby operational decisions are delegated as much as possible to those who have the most detailed knowledge of the consequences and practicalities associated with those decisions.

See Also: Emergence, Inspect and Adapt

References: Wikipedia: Self-Organization, Wikipedia: Complex Adaptive Systems

Return To Glossary

19.11 * Shu

See Also: Shu-Ha-Ri, Ha, Ri

Return To Glossary

19.12 * Shu-Ha-Ri

See Also: Shu, Ha, Ri

Return To Glossary

19.13 * Sidebar

See Also: Parking Lot

Return To Glossary

19.14 * Single Responsibility Principle

See Also: SOLID OOD Principles:

Single Responsibility Principle

Open Closed Principle,

Liskov Substitution Principle,

Interface Segregation Principle,

Dependency Injection Principle

Return To Glossary

* Software Quality Metrics

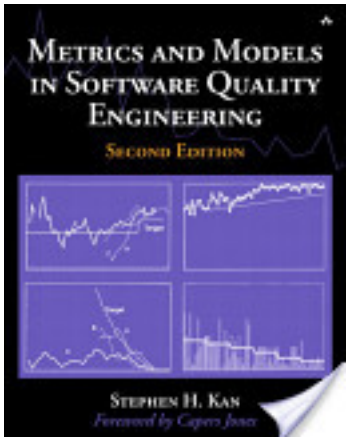
Dynamic Source Analysis

Code Coverage

Line Coverage

Branch Coverage

Method Coverage



Class Coverage

Package Coverage

Static Source Analysis

Coding Standards

Style, Formatting

StyleCop

Architectural Rules Compliance

Findbugs, PMD, Checkstyle, FxCop

LCOM4

Complexity

CCN

NCSS

Coupling

Afferent Coupling

Efferent Coupling

Spider Graph

Duplication: CPD

See Also: Wikipedia, Sonar-Metrics

Return To Glossary

19.15 * SOLID OOD Principles

See Also: Single Responsibility Principle

Open Closed Principle,

Liskov Substitution Principle,

Interface Segregation Principle,

Dependency Injection Principle

Return To Glossary

19.16 Spike

A story or task aimed at answering a question or gathering information, rather than implementing product features, user stories, or requirements. Sometimes a user story is generated that cannot be estimated until the development team does some actual work to resolve a technical question or a design problem. The solution is to create a "spike," which is a story whose purpose is to provide the answer or solution. Like any other story or task, the spike is then given an estimate and included in the sprint backlog.

Return To Glossary

19.17 Sprint

The Scrum term for an iteration. The sprint starts with a sprint planning meeting. At the end of the sprint there is a sprint review meeting, followed by a sprint retrospective meeting.

References: Wikipedia

Return To Glossary

19.18 Sprint Backlog

A list of features, user stories or tasks that are pulled from the product backlog for consideration for completion during the upcoming sprint. Product backlog features and user stories are broken down into tasks to form the sprint backlog during the sprint planning meeting.

See Also: Backlog, Sprint

References: Wikipedia

Return To Glossary

19.19 Sprint Burn-Down Chart

See Burn-Down Chart

19.20 Sprint Planning Meeting

Each sprint begins with a two-part sprint planning meeting, the activity that prioritizes and identifies stories and concrete tasks for the next sprint. For a one-month or four-week sprint, this two-part meeting

should last eight hours; for a two-week sprint, it lasts about four hours. As a general rule of thumb, the number of weeks in a sprint multiplied by two hours equals the total length of the spring planning meeting.

a. Part one of the sprint planning meeting is a review of the product backlog. This is when the product owner describes what needs to be built for the next sprint. During this part of the meeting, it is not uncommon for the team to discuss the sprint objectives with the product owner, and ask clarifying questions and remove ambiguity.
b. During part two of the sprint planning meeting, the team decides how the work will be built. The team will begin decomposing the product backlog items into work tasks and estimating these in hours. The product owner must be available during this meeting but does not have to be in the room. The output of the second planning meeting is the Sprint Backlog.

References: Wikipedia, Card-Conversation-Confirmation

Return To Glossary

19.21 Sprint Review

A meeting held at the end of each sprint in which the delivery team shows what they accomplished during the sprint; typically this takes the form of a demo of the new features. The sprint review meeting is intentionally kept very informal. With limited time allocated for Sprint review prep. A sprint review meeting should not become a distraction or significant detour for the team; rather, it should be a natural result of the sprint.

References: Wikipedia

Return To Glossary

19.22 Stakeholder

Anyone external to the team with a vested interest in the outcome of the team's work.

See Also: Chicken

References: Wikipedia

Return To Glossary

19.23 Standup Meeting

The Daily Standup Meeting is a minimalist status meeting, time-boxed to fifteen minutes. Its purpose is to ensure that questions are answered quickly, that issues are identified and addressed quickly, and to provide Team members with a common understanding of how the Sprint is progressing. The ScrumMaster facilitates this meeting.

Three questions asked:

i. What have you done since last daily scrum?
ii. What will you do before the next daily scrum?
iii. What obstacles are impeding your work?

These items are often referred to as Y-T-I (Yesterday, Today, Impediments)

The ScrumMaster ensures that participants call sidebar meetings for any discussions that go too far outside these constraints.

The Scrum literature recommends that this meeting take place first thing in the morning, as soon as all team members arrive.

a.k.a: Daily Scrum, Daily Standup

References: Wikipedia

Return To Glossary

19.24 Story

Scrum requirements written in short narrative form. There are 5 types of requirement stories:

1. User Story
2. Technical Story
3. Defect
4. Spike
5. Tracer Bullet

See User Story

References: Wikipedia, Card-Conversation-Confirmation

Return To Glossary

19.25 Story Points

Story points are an abstract measure of effort to implement a story. Story points can be evaluated in either Absolute or Relative units.

Absolute units are directly related to time with 1 story point equal to 8 person hours of work. Because absolute units are directly related to time they can be compared across teams.

Relative units are based on a known pivot story and are rated as either larger or smaller than the pivot by some factor. The size of the pivot is specific to the team, therefore estimates of stories across teams are not equal, nor are velocity measurements.

See also: Estimation

Return To Glossary

19.26 * Stress Test

See Also: Load Test

Return To Glossary

19.27 * Swarming

Return To Glossary

20 T

20.1 Task

Tasks are descriptions of the actual work that an individual or pair does in order to complete a story. They are manageable, doable, and trackable units of work. Typically, there are several tasks per story. Tasks have the following attributes, and all tasks must be verified complete - not just "done":

i. A description of the work to be performed, in either technical or business terms
ii. An estimate of how much time the work will take (hours, days)

iii. An owner, who may or may not be pre-assigned
iv. An exit criteria and verification method (test or inspection)
v. An indication of who will be responsible for the verification

See Also: Story, Sprint Planning, Task Breakdown

References: Wikipedia

Return To Glossary

20.2 Task Board

A chart that presents, at minimum, "to do", "in progress", and "done" columns for organizing a team's work. Some teams include their backlog as a column on the task board, while others limit it to work to be performed during the current iteration. Ideally, the task board is a physical thing, consisting of note cards or sticky notes affixed to a wall, although distributed teams may use an online task board application. The task board may illustrate tasks or other forms of work such as user stories. In Scrum, the task board is often used to illustrate the tasks for the current sprint, populated with tasks for the current sprint, while other Agile teams may populate it with user stories.

See Also: Sprint Backlog, Sprint Planning, Task, Kanban, Big Visible Charts

References: Wikipedia

Return To Glossary

20.3 * Task Breakdown

Return To Glossary

20.4 Team

In Agile Software Development, the team refers to the cross-functional group of people that have made a collective commitment to work together to produce the work product and improve their performance over time. In addition to software development and test roles, the team may include any skill set necessary to deliver the work product.

In Scrum the Team can refer to one of two groups of people

1. Scrum Team: members identified by each of 3 Scrum roles.
2. Delivery Team: A cross-functional subset of the Scrum Team.

See Also: Scrum Team, Delivery Team, Self-Organization

Return To Glossary

20.5 Technical Debt

A term coined by Ward Cunningham to describe the obligation that a software organization incurs when it chooses a design or construction approach that's expedient in the short term but that increases complexity and is more costly in the long term. Whether or not to incur technical debt is a tradeoff decision that ideally is made in a deliberate manner at the point that work occurs.

See Also: Refactoring

References: Wikipedia

Return To Glossary

20.6 * Technical Story

See Also: Story

Return To Glossary

20.7 Test Automation

"The use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting function." (Wikipedia)

In agile development, test automation is frequently used to automate unit tests, integration tests, and functional tests. Since the definition of done for most agile projects requires that code be thoroughly tested by the end of the iteration, test automation is critical if not necessary to obtain acceptable velocity. In addition, for most practical purposes, test automation is necessary to effectively apply continuous integration and remain true to the commitment to not "break the build."

See Also: Unit Testing

References: Wikipedia

Return To Glossary

20.8 Test-Driven Development

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

Ken Beck is credited for having invented TDD, one of the original 12 XP practices.

See Also: Unit Testing

References: Wikipedia

Return To Glossary

20.9 Time-box

A time-box is a time period of fixed length allocated to achieve some objective. In agile development, iterations and sprints are examples of time-boxes that limit work in process and stage incremental progress. Time-boxes are often used to avoid over-investing in tasks such as estimating development tasks.

References: Wikipedia

Return To Glossary

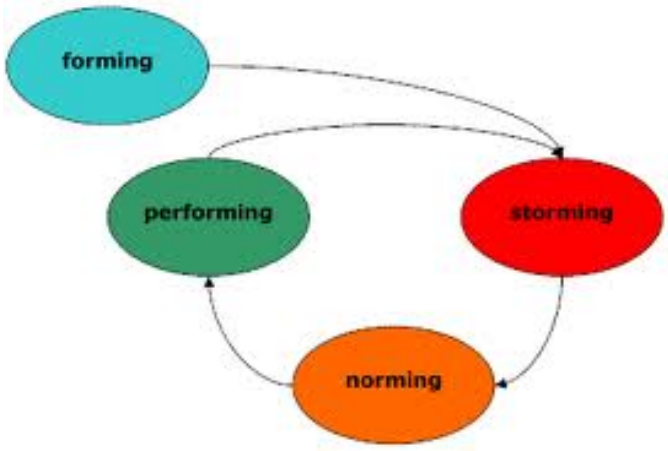
20.10 * Tracer Bullet

Return To Glossary

20.11 * Transparency

Return To Glossary

20.12 * Tuckman Model



References: Wikipedia, TeamTechnology, TuckmansTeamDevelopmentModel.pdf

Return To Glossary

21 U

21.1 Unit Testing

"A unit is the smallest testable part of a software system. In procedural programming, a unit may be an individual function or procedure." (Wikipedia)

Comprehensive unit test coverage is an important part of software integrity and should be automated to support the incremental delivery requirements of agile software development teams. In most cases, unit testing is the responsibility of the developer.

See Also: Test-Driven Development, Test Automation

References: Wikipedia

Return To Glossary

21.2 User Story

A requirement, feature and/or unit of business value that can be estimated and tested. Stories describe work that must be done to create and deliver a feature for a product. Stories are the basic unit of

communication, planning, and negotiation between the Scrum Team, Business Owners, and the Product Owner. Stories consist of the following elements:

i. A description, usually in business terms
ii. A size, for rough estimation purposes,
    1. generally expressed in story points (such as 1, 2, 3, 5)
iii. An acceptance test, giving a short description of how the story will be validated

See Also: Story, Technical Story, Defect, Spike, Tracer Bullet, INVEST

References: Wikipedia

Return To Glossary

22 V

22.1 Velocity

Velocity measures how much work a team can complete in an iteration. Velocity is often measured in stories or story points. Velocity may also measure tasks in hours or an equivalent unit. Velocity is used to measure how long it will take a particular team to deliver future outcomes by extrapolating on the basis of its prior performance. This works in Agile development, when work is comprehensively completed after each iteration.

References: Wikipedia

Return To Glossary

22.2 * Velocity Tracking

Return To Glossary

22.3 Vision

See Product Vision

Return To Glossary

22.4 Voice of the Customer (VOC)

"Voice of the Customer (VOC) is a term used in business and Information Technology (through ITIL) to describe the in-depth process of capturing a customer's expectations, preferences, and aversions. Specifically, the Voice of the Customer is a market research technique that produces a detailed set of customer wants and needs, organized into a hierarchical structure, and then prioritized in terms of relative importance and satisfaction with current alternatives." (Wikipedia)

References: Wikipedia

Return To Glossary

23 W

23.1 Wiki

An editable intranet site where details of stories and tracking information may be recorded during development.

References: Wikipedia

Return To Glossary

23.2 * WIP

See Work inProgress.

Return To Glossary

23.3 * Work Breakdown Structure

Return To Glossary

23.4 Work in Progress (WIP)

Any work that has not been completed but that has already incurred a capital cost to the organization. Any software that has been developed but not deployed to production can be considered a work in progress.

References: Wikipedia

Return To Glossary

24 X

24.1 XP

See: Extreme Programming

Return To Glossary

25 Y

25.1 * YAGNI

You Aint Gonna Need It (yet)

Return To Glossary

25.2 * YAGRI

You Aint Gonna Release It (yet)

Return To Glossary

26 Z

26.1 Reference Articles and Papers

The following are links and abstracts relating to various articles and blogs found on the web. This collection of articles were chosen for their value and importance to Agile Software Development.

Agile Architectures

Return To Glossary

26.1.1 Agile Architecture

by Chris Sterling @ SolutionsIQ, CST

by Mickey Phoenix @ SolutionsIQ, CSM

As companies begin to embrace Agile methods, questions about architecture begin to emerge. In this presentation, learn about the approaches two experts took to better align businesses with architecture goals.

Distributed Scrum

Return To Glossary

26.1.2 Successful Distributed Agile Team Working Patterns

by Monica Yap @ SolutionsIQ, CSM

Explore some common successful distributed team working patterns that have been used on distributed Agile development projects in this white paper and related presentation.

26.1.3 Case Study: Implementing Distributed Extreme Programming

by Monica Yap @ SolutionsIQ, CSM

This white paper details the challenges a team at WDSGlobal faced in a distributed development environment, lessons learned, and how issues such as global continuous integration, cultural differences, and conflicting priorities were resolved across regions.

26.1.4 Daily Scrums in a Distributed World

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

Meta-Scrum

Return To Glossary

26.1.5 Establishing and Maintaining Top to Bottom Transparency Using Meta-Scrum

by Brent Barton @ SolutionsIQ, CST

Learn how a properly executed Meta-Scrum helps drive transparency vertically into the organization in this Agile Journal article.

Agile Adoption

Return To Glossary

26.1.6 Introduction to Scrum

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

The benefits and practices of Scrum

26.1.7 Scrum as Project Management

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

Comparing and Contrasting Scrum to Traditional Project Management

26.1.8 The Agile Story: Scrum Meets PMP

by Crystal Lee @ cPrime, PMP, CSM

Do you know what a Scrum is? Wondering if you should try Scrum on your next project?

26.1.9 When to Use Scrum

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

Scrum is a lightweight agile process framework used primarily for managing software development.

26.1.10 Agile Top-Down: Striking a Balance

by Bryan Stallings @ SolutionsIQ, CST

Agile is being evangelized in executive boardrooms and introduced top-down with increasing frequency. Learn about the appropriate role of senior leadership in an effective Agile transformation in this Agile Journal article.

26.1.11 Agile ROI Part I: The Business Case for Agility

by John Rudd @ SolutionsIQ

This Agile Journal article describes some of the financial benefits of adopting Agile and how to quantify the potential value of these innovative practices for your organization. Learn how Agile methods can help financial professionals squeeze money out of work-in-process, drive risk out of projects, and improve project and portfolio return.

26.1.12 Agile ROI Part II: The Business Case for Agility

by David Wylie @ SolutionsIQ

This presentation explores how to quantify the potential value of Agile practices for your organization and how to demonstrate this value for key decision makers.

26.1.13 Scrum in the Enterprise

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

"Scrum in the Enterprise" is a white paper written by Kevin Thompson, one of cPrime's Agile Implementation Specialists. The paper talks about the common issues that companies face while making the transition to Agile Development, while explaining how to prepare for and overcome them. Kevin writes about topics from how to use Scrum in a hybrid environment to how to collaborate in Scrum teams. This white paper will interest and benefit anyone who is involved with Agile projects or just interested in the methodology.

26.1.14 How Uncertainty Works

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

Why it exists, how it behaves, how it accumulates, how to reduce it, and how to cope with it.

26.1.15 The Price of Uncertainty

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

A Mathematical Analysis of Classic and Agile Processes

Proponents of agile development processes, such as Scrum, frequently claim that agile projects are more likely to be successful than traditional plan-driven projects. Unfortunately, attempts to validate this claim based on statistical evidence are difficult. The difficulty arises partly because the two approaches have different concepts of success, and partly because definitions of success

are not uniform even within each approach. This paper addresses the question by performing a simple mathematical analysis of plan-driven and agile projects.

26.1.16 How Agile should your Project be?

by Kevin Thompson @ cPrime.com, CSM, CSP, PMP, PhD

Advocates of agile development claim that agile software projects succeed more often than classic plan-driven projects. Unfortunately, attempts to validate this claim statistically are problematic, because "success" is not defined consistently across studies. This paper addresses the question through a mathematical analysis of these projects. We model agile and plan-driven software projects with identical requirements, and show how they are affected by the same set of unanticipated problems. We find that that the agile project provides clear benefits for return-on-investment and risk reduction, compared to the plan-driven project, when uncertainty is high. When uncertainty is low, plan-driven projects are more cost-effective. Finally, we provide criteria for choosing effective process types.

26.1.17 Integrating Waterfall and Agile Development

by Shayan Alam @ cPrime.com, PMP

Use these tips to help integrate both methodologies into your development organization.

26.1.18 Rational Unified Process Best Practices

by Crystal Lee @ cPrime.com, PMP,

Provide background on each best practice, in the context of current RUP adoption.

26.1.19 Effective Retrospectives

by Kendrick Burson @ cPrime.com, CSM, CSPO

A better understanding of Team Retrospectives with plenty of examples of different patterns for facilitating.

26.1.20 Transitioning From Time-Based to Relative Estimation

by Ilan Goldstein @ ScrumAlliance.org, CSM, CSPO, CSP

Congratulations! You've finally convinced the team that relative story point estimation is a great way to move forward and you're now ready to jump into your first planning poker session. So

where do you start? What's a 1-point story? What's a 3-point story? What's a 13-point story? Your team is looking to you and this process is almost as new to you as it is to them.

Most of the issues with gathering requirements in agile software development and agile testing derive from issues with User Stories. Somehow expressing requirements in such a simple form causes a lot of trouble to agile teams. Of course art of writing good User Stories is the most difficult for new teams starting with a new agile project or these, which freshly transformed development methods to agile software development methodologies. Mistakes made at that point lead to wrong Test Cases, wrong understanding of requirements, and the worst of all wrong implementation which can be direct cause of rejecting the deliverables of the iteration. Lets take a look at the five most common mistakes people make writing User Stories.

26.1.215 Common Mistakes We Make Writing User Stories

by Krystian Kaczor @ ScrumAlliance; CSM, CSP

Most of the issues with gathering requirements in agile software development and agile testing derive from issues with User Stories. Somehow expressing requirements in such a simple form causes a lot of trouble to agile teams. Of course art of writing good User Stories is the most difficult for new teams starting with a new agile project or these, which freshly transformed development methods to agile software development methodologies. Mistakes made at that point lead to wrong Test Cases, wrong understanding of requirements, and the worst of all wrong implementation which can be direct cause of rejecting the deliverables of the iteration. Lets take a look at the five most common mistakes people make writing User Stories.

26.1.22 Agile Project Dashboards

Bringing value to stakeholders and top management

by Leopoldo Simini @ ScrumAlliance; CSM, CSP

"Scrum is all about delighting customers and delivering value to stakeholders." I have read this kind of statement since my first day working with Scrum in 2007. Even more, I've had the privilege of taking part on Scrum teams th...

26.1.23 Daily Stand-up, Beyond Mechanics: A Measure of Self-Organization

by Bachan Anand CSM, CSPO, CSP

26.1.24 Affinity Estimation for Release Planning

by Monica Yap @ SolutionsIQ

26.1.25 Managing Risk in Scrum, Part 1

by Valerie Morris @ SolutionsIQ

26.1.26 Product Owner Anti-Patterns

by Monica Yap @ SolutionsIQ

Part 1: The Absent Product Owner

Part 2: The Churning Backlog

Part 3: No Single Product Owner

Part 4: Copy the Old One

26.1.27 Card-Conversation-Confirmation

by Ron Jeffries, 2001

XP Practices for generating a well groomed backlog, elaborating story contents and validating completed results.

"User stories have three critical aspects. We can call these Card, Conversation, and Confirmation."

Ron Jeffries, 2001

26.1.28 Recognizeing Bottlenecks in Scrum

by Dhaval Panchal @ SolutionsIQ, CST

Part 1

Part 2

26.1.29 If At First You Don't Succeed, Fail, Fail Again

by Michael Tardiff @ SolutionsIQ, CSM, CSPO

26.1.30 What is the Definition of Done (DoD) in Agile?

by Dhaval Panchal @ SolutionsIQ, CST

DoD is a collection of valuable deliverables required to produce software.

DoD is the primary reporting mechanism for team members.

DoD is informed by reality.

DoD is not static

DoD is an audit-able checklist.

26.1.31 How Should We Deal With the Mess That Scrum Exposes?

by Monica Yap @ SolutionsIQ, CSM, CSPO

Part 1 of 5) How Should We Deal With the Mess That Scrum Exposes?

Part 2 of 5) Scrum Exposes the Mess With No Quality Built In

Part 3 of 5) Scrum Exposes the Mess of Unstable Code Base

Part 4 of 5) Scrum Exposes the Mess of Excess Specialists

Part 5 of 5) The Mess That Scrum Exposes: Putting It All Together

26.1.32 The Afternoon ScrumMaster: Keeping Agile Teams on Track

by Dhaval Panchal @ SolutionsIQ

26.1.33 The Short Short Story

by Paul Dupuy @ ScrumAlliance; CSM

The short short story: How long does it have to be? Scrum teams often use user stories for backlog items. Unfortunately, one of the most important aspects of a story—its extremely short length—has been subtly transformed over time, an…

26.1.34 Is Sustainable Pace Nice to Have? Think Again!

by Manoj Vadakkan CSM, CSP

Most of the time, "selling" Agile is easy these days. Everyone agrees that iterative and incremental development is a better alternative; more user interaction is better; so on and so forth. At some point, I will talk about the import...

26.1.35 Agile User Interface Design and Information Architecture From the Trenches

by Robin Dymond @ ScrumAlliance; CSM, CSP, CST

I was a Technology Director in a large web design company 6 years ago, and they failed to adopt Scrum. There were numerous management dysfunctions; however the Creative managers were the most resistant. Primarily, it was a case of not wanting real...

26.1.36 Why Agile Does Matter in an Embedded Development Environment

by Bent Myllerup @ ScrumAlliance; CSM, CSPO, CSP, CSC

The software industry has achieved great results by introducing agile methods like Scrum. Agile methods create outcomes that benefit customers as well as management and employees of the business. The results have been proven in the form of increas...

26.1.37 The Illusion of Precision

by Jim Schiel @ ScrumAlliance; CSM, CSP, CST

For me, one of the most intriguing, yet not explicitly stated, fundamentals of AgileDevelopment is the practice of analyzing and designing just enough of what we are planning to build that we can then move forward to build it. You can find specifi...

26.1.38 Specialization and Generalization in Teams

by Bas Vodde @ ScrumAlliance; CSM, CSPO, CSP, CST

Specialization in Scrum has been a hot topic for many years and pops up at every Scrum course I run. It is an important issue that's particularly relevant for a new team in their first Sprint. Scrum defines specialization as a cross-functio...

26.1.39 The Importance of Self-Organisation

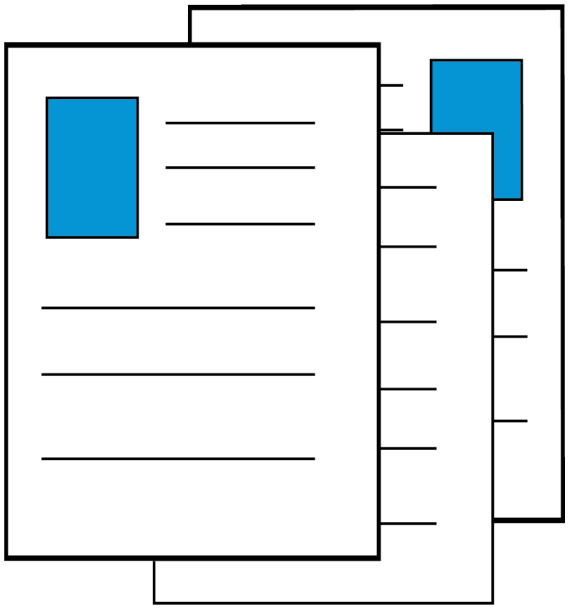
by Geoff Watts @ ScrumAlliance; CSM, CSP, CSC, CST

> "An empowered organization is one in which individuals have the knowledge, skill, desire, and opportunity to personally succeed in a way that leads to collective organizational success." -- Stephen R. Covey, Principle-centered Leadership

26.1.40 Manager 2.0: The Role of the Manager in Scrum

by Pete Deemer @ ScrumAlliance; CSM, CSP, CST

Agile/Scrum Glossary of Terms



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

Agile Estimation:

Agile estimation is a process of agreeing on a size measurement for the stories in a product backlog. Agile estimation is done by the team, usually using Planning Poker.

The Agile Manifesto:

The Agile Manifesto was developed by a group fourteen leading figures in the software industry, and reflects their
experience of what approaches do and do not work for software development.

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

Agile Methodology:

Agile Methodology is an umbrella term for several iterative and incremental software development methodologies.
The most popular agile methodologies include: extreme programming (XP), Scrum, Crystal, Dynamic Systems
Development (DSDM), Lean Development, and Feature Driven Development (FDD). All Agile methods
share a common vision and core values of the Agile Manifesto.

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 Planning Basics:
The four basics of Agile planning are: Product Backlog, Estimates, Priorities and Velocity.

Agile Planning - Estimates:
Estimates answer the question: "How long will it take or how many can we do by a given date?"

Agile Planning – Priorities:
Priorities answer the question: "Which capabilities are most important?"
Agile Planning - Product Backlog:
The Product Backlog answers the question: "What capabilities are needs for financial success?"

Agile Planning – Velocity:
Velocity answers the question: "How much can the team complete in a Sprint?"

Agile Software development:
Agile software development is a group of software development methodologies based on interative and incremental development, where requirements and solutions evolve through collaboration between self organizing team, cross functional teams.

Burndown Chart:
A Burn down chart is a chart showing how much work remaining in a sprint. Calculated in hours remaining and maintained by the Scrum Master daily. Business Value: Each user story in the Product Backlog should have a corresponding business value assigned. Typically assign (L,M,H) Low, Medium, High. Product Owner prioritizes Backlog items by highest value.

Capacity:
Capacity is the Number of Teammates (Productive Hours x Sprint Days). Example: Team size is 4, Productive hours are 5, Sprint length is 30 days. Capacity = 4(5x30) = 600 hours.

Chickens:
Chickens are the people that are not committed to the project and are not accountable for deliverables.

Daily Scrum Meeting:
The Daily Scrum Meeting is a minimalist status meeting, time-boxed to fifteen minutes. Its purpose is to ensure that questions are answered quickly, that issues are identified and addressed quickly,

and to provide Team members with a common understanding of how the Sprint is progressing. The ScrumMaster facilitates this meeting. Three questions asked: "What have you done since last daily scrum?" "What will you do before the next daily scrum?" "What obstacles are impeding your work?"

Defect:
A defect is a failure or bug of the product to behave in the expected fashion. Defects are stored in a bug-tracking system, which may or may not be physically the same system used to store the Product Backlog. If not, then someone (usually the Product Owner) must enter each Defect into the Product Backlog, for sequencing and scheduling.

Done:
The term Done is used to describe a product increment that is considered releasable; it means that all design, coding, testing and documentation have been completed and the increment is fully integrated into the system.

Epic:
An Epic is a very large user story that is eventually broken down into smaller stories; epics are often used as place holders for new ideas that have not been thought out fully. Impediment: Anything that prevents the team from meeting their potential. If organizational, it is the Scrum Master's responsibility to eliminate it. If it is internal to the team, then they themselves should deal with it.
INVEST criteria for User Stories:
INVEST: Independent, Negotiable, Valuable, Estimatible, Small, Testable.

Pigs:
Pigs are the people who are accountable for project's success.

Planning Poker:
Planning poker is a game used to apply estimates to stories. It uses a voting approach designed to avoid influence bias.

- How it Works:
    1. Each estimator selects a set of cards.
    2. Facilitator reads item to be estimated, and moderates a brief discussion to clarify details.
    3. Facilitator calls for estimates. Each estimator places estimate face down, hiding the value.
    4. Facilitator calls for vote, and all estimators turn over cards at the same time.

5. If all cards agree, their value is recorded as the estimate.
6. Otherwise, facilitator asks high and low estimators to explain their reasoning, and moderates a brief discussion to clarify issues.
7. Repeat 3-6 until estimates converge.

Product Backlog:
The Product Backlog is the set of all un-implemented Product Backlog Items (requirements, in the form of Stories and Defects) that have not been assigned to the current Sprint. Unlike the Sprint Backlog, there is no requirement that all PBIs be assigned a rank.

Product Owner:
The Product Owner is the keeper of the requirements. He provides the "single source of truth" for the Team regarding requirements and their planned order of implementation.

In practice, the Product Owner is the interface between the business, the customers, and their product related needs on one side, and the Team on the other. He buffers the Team from feature and bug-fix requests that come from many sources, and is the single point of contact for all questions about product requirements. He works closely with the team to define the user-facing and technical requirements, to document the requirements as needed, and
to determine the order of their implementation. He maintains the Product Backlog (which is the repository for all of this information), keeping it up to date and at the level of detail and quality the Team requires.
The Product Owner also sets the schedule for releasing completed work to customers, and makes the final call as to whether implementations have the features and quality required for release.

Release Backlog:
The Release Backlog is the same as the product backlog. May involve one or more sprints dependent on determined release date.

Retrospective Meeting:
The Retrospective meeting is held after the Sprint Review meeting. Its purpose is to provide the Team an opportunity to learn, and therefore improve, from the experience of the just-concluded Sprint. It answers the following questions:

- "What worked well, that we should do again?"
- "What didn't work well?"
- "What changes we should make for next time?"

.

Scrum Artifacts:

The three Scrum Artifacts are the Sprint Backlog, the Product backlog, and the Burndown chart.
Scrum Development:

Scrum Meetings:
The three Scrum meetings are the Sprint planning meeting, the Daily Scrum Meeting, Sprint Review meeting, and the Retrospective meeting. Each time box has a specified start and duration, and work is not allowed to extend beyond the duration.

Scrum Process:
Sprint planning --> Product backlog --> Sprint Backlog --> Daily Scrum --> Sprint --> Shippable Product --> Sprint Retrospective --> Product Backlog... etc

Scrum:
Scrum is a lightweight process framework for agile development, and the most widely-used one.

ScrumMaster:
The Scrum Master is the keeper of the process. He/she is responsible for making the process run smoothly, for removing obstacles that impact productivity, and for organizing and facilitating the critical meetings.

Self-Organizing:
Teams are self-organized meaning they have both responsibility & authority. They are all motivated by a common goal.

Single Wringable Neck:
This is the Product Owner.

Sprint Backlog:
The Sprint Backlog is the set of Product Backlog Items, or PBIs (Stories and Defects) planned for implementation in a Sprint. The items in the Sprint Backlog must be ranked in the desired order of implementation (a Product Owner responsibility). The ranking reflects both the urgency (value) of the item, and any dependencies that exist between items.

Sprint Planning Meeting:
The ScrumMaster facilitates the Sprint Planning Meeting, which kicks off the Sprint, and which is attended by the team members and the Product Owner. The purpose of this meeting is to select from the Product Backlog those Product Backlog Items (PBI's) the Team intends to implement in this Sprint.

Sprint Review Meeting:
The Spring Review meeting (also called Sprint Demo) held at the end of the Sprint. The team demonstrates the Sprint's completed Product Backlog, to the Product Owner and other interested parties. This meeting gives the Product Owner a final chance to make a go/no-go release decision, and gives the Team members a chance to show off their work.

Sprint Task:
A Sprint Task is a single small item of work that helps one particular story reach completion.

Sprint:
the basic development cycle for a project. Most often a sprint is 2-4 weeks, however, different organizations and projects select Sprint lengths that best meet their needs.

Stakeholders:
Anyone who needs something from the team or anyone who the team needs something from. Examples: Executives, Auditors, Security specialists, Enterprise Architects, Support engineers etc.

Story points:
A Story Point is a simple way to initially estimate level of effort expected to develop. Story points are a relative measure of feature difficulty. Usually scored on a scale of 1-10. 1=very easy through 10=very difficult. Example: "Send to a Friend" = 2, "Shopping Cart"=9

Story Template:
A Story Template looks like this: "As a "user" I want "function" so that "result". Example: As a user, I want to print a recipe so that I can cook it.

Task Board:
A task board is a white board containing the teams Sprint goals, backlog items, tasks, tasks in progress, "DONE" items and the daily Sprint Burndown Chart. The Scrum meeting is best held around task board & visible to everyone.

Team:
The Team is a self-organized and cross functional group of people who do the hands-on work of developing and testing the product. They are responsible for producing the product, it must also have the authority to make decisions about how to perform the work. Team is 7+/-2 members.

Technical Story:
Technical stories are the requirements that do not represent user-facing features, but do represent significant work that may support user-facing features.

Three Scrum Roles:
The three Scrum roles are the ScrumMaster, the Product Owner & the Team.

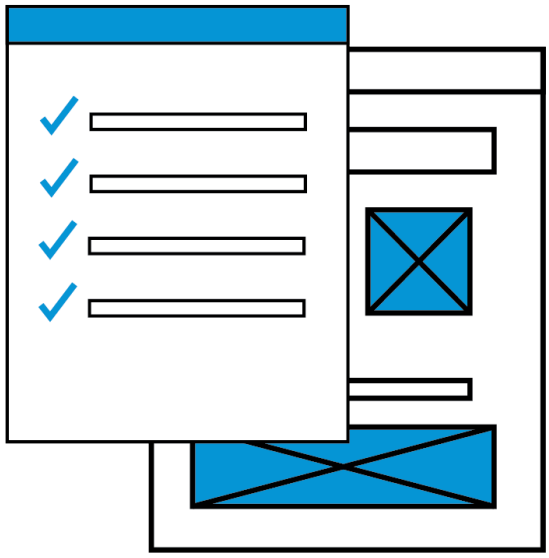
Time box:
A Time box is a span of time of fixed duration, dedicated to a particular purpose, whose boundaries are strictly enforced. Scrum defines several "official" time boxes, such as the Sprint and critical meetings, but the concept of a time box is an important theme that permeates scrum projects.

User Story:
A User story describes a desired feature (functional requirement) in narrative form. It is usually contains a name, description, screen and external documents, and information about how the implementation will be tested. User stories are usually written by the Product Owner, and are the Product Owner's responsibility.

Velocity:
Velocity is the rate at which team converts items to "DONE" in a single Sprint. – Usually calculated in Story Points.

Scrum Team Member Cheat Sheet

A quick list of the actions a Scrum Team Member performs.



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

Every Day

Start of Day

1. Look at the Burndown chart for this Sprint.
    a. If the Team is running behind schedule, see if you can help to bring it back on track (especially if one of your tasks is running slow). Make sure that tasks you have completed have a Completed status, so they contribute to the progress.

2. Review the tasks you'll be working on today.
    a. If you don't have a list of tasks for the day, work with your team members to figure out who will be doing the day's tasks. Always plan to work on the Sprint Backlog Items in rank order.

During the Day

1. When you start or finish a task, update its status immediately.
    a. Find the item you are working on, and update the relevant fields.
        If you are starting a task, change its status to In Progress & assign owner
        If the task is done, change its Status to Completed.
        Update To-Do with the hours remaining (zero, if the task is Completed).

    b. Check the Total To-Do value for the Sprint, before and after your changes, to confirm that your status change reduces the amount of work remaining.

2. Work on your tasks. If you need help, get it right away! Don't wait.

3. Collaborate! Talk to anyone you need to talk to in order to get the work done.

4. Attend the Daily Scrum meeting.

    a. Provide your information:
        What I've done since the last meeting.
        What I plan to do before the next meeting.
        What issues are slowing me down, for which I need help.

    b. See if you can help someone who needs help.

5. Assist Product Owner with requirements development.
    a. Answer questions, provide insight

6. Write Technical Stories to address future, non-user-facing requirements

7. Write Defects for any bugs you find that are not addressed by acceptance tests for the Story on which you are working

8. Demonstrate each Sprint Backlog Item to the Product Owner as early as possible. If the implementation is not as the Product Owner desires, then the Product Owner will make the call about what to make changes now, or whether to defer them to a future Sprint, as a new Story.

End of Day

1. Update the status of any tasks that need to be updated.

a. If you are tracking work remaining (To Do) for tasks that are in process, update the field

2. Look at the Burndown chart to see how the Team is doing on this Sprint.

To Prepare for the next Sprint Planning Meeting

Prepare the Product Backlog before the next scheduled Sprint Planning meeting, through Backlog Grooming meetings. These meetings are commonly held three times per Sprint. The Product Owner provides set of draft Backlog Items and Epics your review before each of these meetings, and you should read them and prepare your questions and comments.

1. In the first Backlog Grooming meeting, give feedback to the Product Owner. Identify new Stories that should be written to fill 'holes,' including Technical Stories that the Team must write and implement to enable implementation of User Stories.

After the meeting

a. Write any Technical Stories required.

2. In the second Backlog Grooming meeting, give feedback to the Product Owner. Work with the Product Owner to define ranking of the Backlog Items, especially with respect to dependencies. Identify any further revisions required.

a. Revise Technical Stories, as needed.

3. In the third Backlog Grooming meeting, review revisions and agree on finalized Backlog Items and their ranking.

Just before the Sprint Planning Meeting

Prepare for the Sprint Planning meeting. Do this at the scheduled time before the Sprint Planning Meeting.

1. Bring a list of unresolved issues and questions about the candidate Product Backlog Items to the Sprint Planning meeting.

In the Sprint Planning Meeting

The ScrumMaster facilitates this meeting, which the Team members and Product Owner attend.

1. Discuss and estimate each Backlog Item.

Immediately after the Sprint Planning Meeting

Create Task Breakdowns as soon as the Sprint Planning meeting ends. It is important to do this work right away, so that the full scope of planned effort is known on the first day start of the Sprint. (Otherwise, tracking becomes confusing.) Product Owners do not normally attend this meeting. ScrumMasters may attend, if it is useful for them to do so.

1. Work with all Team members to create a Task Breakdown for each Story and Defect in the Sprint Backlog, and provide an effort estimate (in person-hours) for each Task.

    a. New Teams often have the whole group meet, and work together to create Task Breakdowns that are consistent across the Backlog Items.

    b. Experienced Teams often have one person create candidate breakdowns for a few Backlog Items, then meet to review and finalize the breakdowns to ensure completeness and consistency.

In the Sprint Review Meeting

The Team members are responsible for demonstrating the completed Sprint Backlog Items to the Product Owner, and for deciding who will do the various parts of the demonstration.

In the Sprint Retrospective Meeting

The ScrumMaster facilitates this meeting, which is attended by the Product Owner and all Team members.

1. The ScrumMaster reviews with the Team the status of work items they selected in the previous Retrospective meeting.

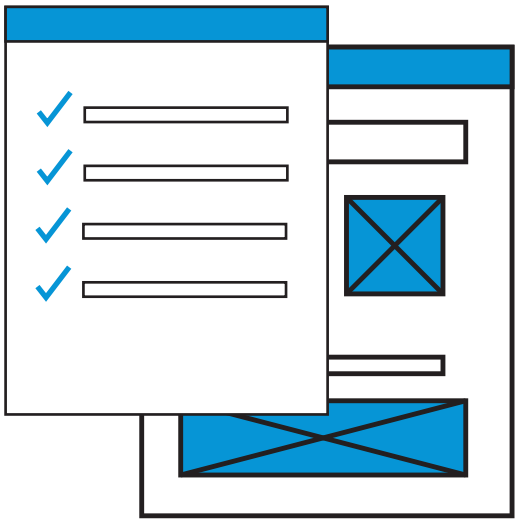
2. The ScrumMaster facilitates information collection, recording what the Team members, Product Owner, and ScrumMaster have to say about

    a. What went well, that we should do again
    b. What didn't go well
    c. Specific suggestions for improvement

3. The ScrumMaster facilitates a discussion to select the top few suggestions to be implemented in the next Sprint, and theTeam members who will own them.

ScrumMaster Cheat Sheet

The easy way to learn your daily tasks as a ScrumMaster



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

Scrum Master Cheat Sheet

Do these things every day.

Start of Day

- [ ] Update and review the current Burndown chart for the current Sprint, for each Team of interest.

  If a Team is running behind schedule, investigate, and try to help bring it back on track. Make sure that tasks that been have completed have a Completed status, so they contribute to the progress.

- [ ] Review the Sprint Backlog Items and their associated tasks.

  Check to see if any tasks have missing information.
  - Missing Estimates for Backlog Items
  - Missing Detail Estimates for tasks
  - Missing To-Do values for tasks
  - Missing Owners for tasks that are In Progress or Complete

  Look for inconsistencies.
  - Backlog Items with status values that can be selected in the tracking tool, but which we have decided not to use.
  - Tasks marked as Complete, which have non-zero To-Do values
  - Tasks with zero To-Do values, which are not marked as Complete
  - Backlog Items marked as Complete, which have any tasks not marked as Complete
  - Backlog Items that have been finished (all of their tasks have been marked as Complete), and should be marked as Complete, but are not

  Follow up with appropriate Team members, to ensure they supply missing information, and correct inconsistencies

During the Day

- [ ] Identify any issues that are impeding progress, which require intervention.

  Assist Team members to resolve issues, as needed.
  - Protect Team members from interference
  - Teach them to solve problems first, and come to you when they can't solve them

- [ ] Facilitate the Daily Scrum meeting.
  - Show the Burndown chart
  - Listen to answers to the standard three questions
  - Ensure follow-up actions and participants are defined
  - Provide any additional information the Team should know

- [ ] Review new User Stories, Technical Stories, and Defects added to the Product Backlog, for clarity and completeness

  Make sure new Product Backlog Items are assigned to the right Teams

End of Day

- [ ] Same as Start of Day: Review status, look for missing or inconsistent information, and follow up with Team members to resolve.

To Prepare for the next Sprint Planning Meeting

- [ ] Facilitate the Backlog Grooming meetings (as per the Product Owner Cheat Sheet).
- [ ] Conduct capacity planning for the next Sprint
    - Gather information about Team member planned time off, holidays, and other impacts to Team Member availability
    - Estimate the Team's Velocity (amount of work Team can do) for the next Sprint
- [ ] Update relevant tools with the Velocity and other Sprint-related information

In the Sprint Planning Meeting

Facilitate the meeting.

- [ ] Go through the top Product Backlog Items, ranked by the Product Owner, and read each one to the Team.
- [ ] Facilitate the estimation process.
- [ ] Enter the estimate for this item into the tool used to record this information.
- [ ] Move the item into the Sprint Backlog.
- [ ] Repeat, and stop when the Sprint is full.
- [ ] (Recommended) Estimate a few more Backlog Items, to expand the pool of pre-estimated Items, for use in future planning meetings (especially Release Planning).

Immediately after Sprint Planning Meeting

- [ ] Immediately after the Sprint Planning Meeting
If the Team needs help with creating Task Breakdowns, the ScrumMaster should facilitate and mentor, as needed. Otherwise, the ScrumMaster's presence is not required (though allowed).

In the Sprint Review Meeting

- [ ] The ScrumMaster has little to do, except for the important part of making sure that the meeting happens. The Team members are responsible for demonstrating the completed Sprint Backlog Items to the Product Owner, and for deciding who will do the various parts of the demonstration.

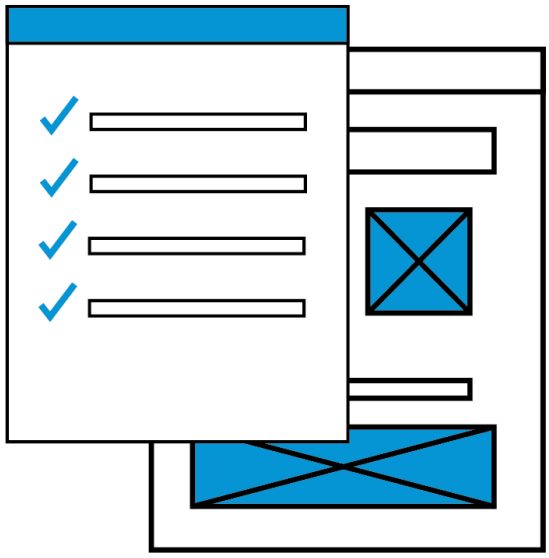
In the Retrospective Meeting

- [ ] The ScrumMaster facilitates this meeting, which is attended by the Product Owner and all Team members.
- [ ] The ScrumMaster reviews with the Team the status of work items they selected in the previous Retrospective meeting.
- [ ] The ScrumMaster facilitates information collection, recording what the Team members, Product Owner, and ScrumMaster have to say about
    - What went well, that we should do again
    - What didn't go well
    - Specific suggestions for improvement

    The ScrumMaster facilitates a discussion to select the top few suggestions to be implemented in the next Sprint, and the Team members who will own them.

Scrum Product Owner Cheat Sheet

A quick list of the actions a Product Owner performs.



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

Every Day

Do these things every day.

Start of Day

1. Look at the Burndown chart for this Sprint of your project.
    a. If you have any concerns about progress, follow up with the ScrumMaster.

During the Day

1. Assist Team members to resolve issues, as needed.
    a. Clarify requirements, answer questions

2. Review the implementation of each Sprint Backlog Item as early as possible (during implementation), to confirm that the result are as desired. If they are not
    a. Decide whether changes should be made now, or if it is preferable to defer them to a future Sprint
    b. If the latter, write a new Story about the changes

3. Write User Stories to address future requirements
    a. Work with Team members to clarify requirements and resolve issues you encounter in writing requirements

4. Write Epics to address future functional requirements too big to fit in a User Story
    a. Create each Epic
    b. Decompose the Epic by adding child Stories or child Epics to it.

5. Write Defects for any bugs you find

6. Attend the Daily Scrum meeting if you and the Team believe this is a productive thing to do.
    a. Listen to answers to the standard three questions
    b. Identify any follow-up actions you need to perform
    c. Provide any additional information the Team should know

To Prepare for the next Sprint Planning Meeting

Prepare the Product Backlog before the next scheduled Sprint Planning meeting, through Backlog Grooming meetings. These meetings are commonly held three times per Sprint.

1. Collect a set of draft Backlog Items and Epics you want the Team to review, and supply to the Team before the first Backlog Grooming meeting. Rank the subset of Backlog Items that you want the Team to implement in the next one or two Sprints.

    a. Drive ranking primarily by value, but also consider risk, penalties for failing to satisfy mandated requirements, and other relevant factors
    b. Include dependencies in the ranking.

2. In the first Backlog Grooming meeting, answer questions and collect feedback from the Team. Identify new Stories that should be written to fill 'holes,' including Technical Stories that the Team must write and implement to enable implementation of your User Stories. After the meeting,

a. Revise Epics and User Stories per Team feedback, and supply to the Team before the next Backlog Grooming meeting.

b. Team Members will write any Technical Stories required.

3. In the second Backlog Grooming meeting, answer questions and collect feedback from the Team during the review of the updat set of Backlog Items. Work with Team to define ranking of the Backlog Items, based on value and dependencies. Identify any furt revisions required.

a. Revise Epics and User Stories per Team feedback, and supply to the Team before the next Backlog Grooming meeting.

b. Team Members will revise Technical Stories, as needed.

4. In the third Backlog Grooming meeting, review revisions and finalize the Backlog Items and their ranking.

In the Sprint Planning Meeting

Work with the Team and ScrumMaster to make the planning meeting productive. Product Owners must attend Sprint Planning meetings.

1. Answer questions to clarify requirements or resolve issues that affect implementation and estimation.
2. Update title and description of Backlog Items to reflect improved understanding, if needed.
3. If desired, revise ranking of reviewed items to make more efficient use of Sprint's capacity (e.g., to fill the Sprint if there would otherwise be unused capacity).

In the Sprint Review Meeting

Reviews the Team's implementations of the Sprint Backlog Items, and note whether the implementations are as desired. If any are not as desired, write new Stories about the changes needed, to be implemented in future Sprints (often the next one).

In the Sprint Retrospective Meeting

The ScrumMaster facilitates this meeting, which is attended by the Product Owner and all Team members.

1. The ScrumMaster reviews with the Team the status of work items they selected in the previous Retrospective meeting.

2. The ScrumMaster facilitates information collection, recording what the Team members, Product Owner, and ScrumMaster ha to say about

a. What went well, that we should do again

b. What didn't go well

c. Specific suggestions for improvement

3. The ScrumMaster facilitates a discussion to select the top few suggestions to be implemented in the next Sprint, and the Team members who will own them.