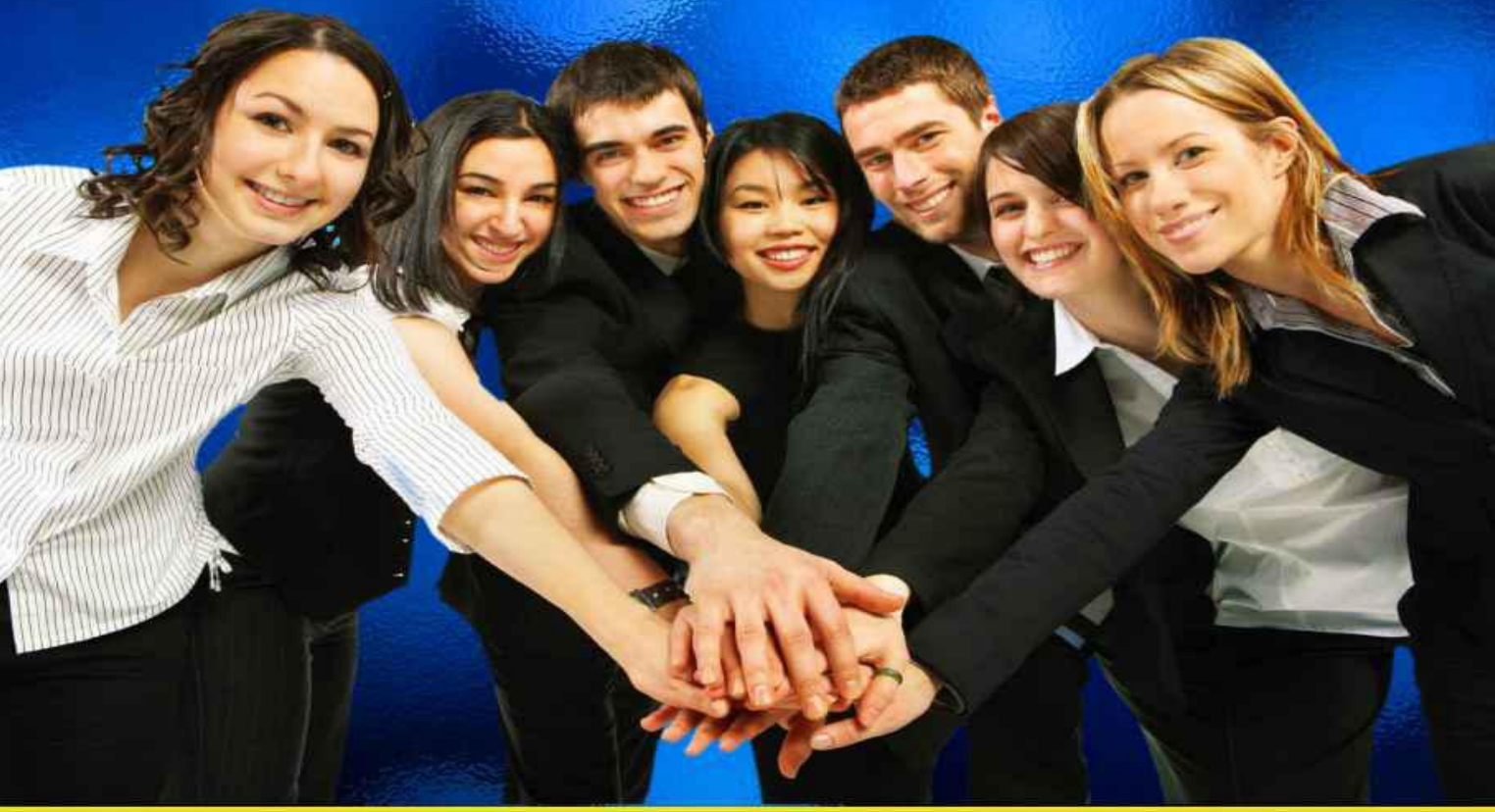
AGILE PROJECT MANAGEMENT FOR BEGINNERS

MASTERING THE BASICS



BRYAN MATHIS

Agile Project Management for Beginners

Mastering the Basics

By Bryan Mathis

Kindle Edition

Disclaimer: This book contains general information that is based on author's own knowledge and experiences. It is published for general reference and is not intended to be a substitute for professional advice. The publisher and the author disclaim any personal liability, either directly or indirectly, for the information

contained within. Although the author and the publisher have made every effort to ensure the accuracy and completeness of the information contained within, we assume no responsibility for errors, inaccuracies, omissions and inconsistencies.

Dedication:

To my parents

Table of Contents

Introduction

My first experience with Agile was in 2005 at a small tech startup company on the West Coast. At the time, the concept of software as a service (SaaS) was in its infancy, and we ended up developing a great product that fell short of its potential because the public was still committed to desktop software.

But the seed had been planted, and I've spent my entire career since then working as a contractor in small startups developing software products. With the emergence of cloud computing in 2008, a lot of pieces fell into place for me. Since then I've been part of several great projects, including a number of mobile web applications. Invariably, the most successful companies I've worked for used Agile to manage their teams.

My intent in writing this book is to use my experience as an Agile team member to introduce

readers to the basic principles of Agile. This is not an advanced Agile practice manual. In fact, I believe that once you learn the basics, the best place to learn advanced Agile methods is on the job. My role is to explain its basic concepts and lay out how the various parts of Agile fit together as a whole system in the real world of project management.

The book's primary focus is on the Scrum variant of Agile because Scrum is the best known and most widely used of the Agile methodologies. In its original form, it predates Agile itself and served as inspiration for Agile's principles. For this reason, I made a conscious decision to use Scrum as a lens through which the reader can view Agile in practice. Readers should know that Scrum isn't the only form of Agile, or even the best one for all projects. It's simply the best vantage point (in my opinion) for understanding Agile, and lightweight software development methods in general.

Agile is a powerful tool for fostering teamwork

and mutual support, but only when all of the team members share the same understanding of how it works. Agile is great for defining roles and generating problem solving strategies, but it can also serve as a source of inspiration and camaraderie. This is especially critical in a startup environment, where the pressure can be tremendous.

The principles of Agile are so powerful that they extend far beyond the software development world. It was my intent in writing this book to present the underlying structure of Agile in simple enough form that you, the reader, can pick it up and apply it to tasks as diverse as managing a non-tech company, building a house, mastering a new skill or hobby, or living a more productive and meaningful life.

I hope this book lays the foundation for you to put the Agile method into practice as you strive for the goals you want to reach, and that it becomes a permanent part of your life toolkit.

- *Bryan Mathis*

Chapter 1: Agile Overview

What Is Agile?

Agile is a lightweight software development method that aims to be more efficient than traditional, plan-driven development models. Agile seeks to do more with less:

- more team-level decision making
- faster development time
- faster response to shifting customer demands
- faster problem solving
- more customer satisfaction
- smaller teams
- less expense
- less wasted effort
- fewer features in the end product that either don't work or are never used

As a contractor on several development teams in the early 2000s, I experienced firsthand the inefficiencies of plan-driven methods for developing software. I confess my guilt as an aider and abettor in developing numerous examples of what came to be known as "bloatware." These programs often took three or more years to develop, yet their customer satisfaction levels were poor. After working on Agile teams for a few years, I came to understand why those massive projects of the past so often turned into massive failures.

Here is a summary of the mistaken assumptions that led to the proliferation of bloatware.

Full-Featured Is Always Best: before Agile, there was a widespread dogma that good customer service always meant developing software with hundreds of features to address each and every perceived customer need, large or small, as if they were of equal importance. In reality, full-featured software was a limited market, and customers

were hungry for simple programs that were easy to learn and would solve their specific problem.

Belief in Static Customer Expectations: another issue was the belief that customer demand could be captured in snapshot form and would conveniently remain unchanged while the development team did its thing. This might have been true before the internet, but as soon as consumers started using the web, customer demand for features began to shift on a daily basis. The old development models were aiming for a target that had long since ceased to exist.

Long Product Development Cycles: predictive (plan-driven) development models dictated that a program couldn't be released until all of its features were finished and bug-free. The desire to release quality products is admirable, but the size and scope of those products meant they would be obsolete long before their release date.

The Agile methodology was a reaction – or maybe

rebellion is a better word – against these dogmas. As a veteran Agile team member, I want to emphasize that this wasn't merely an attempted power grab by coders who didn't like authority. So-called "cowboy coding," where there's no supervision and team members just go off and do whatever strikes their fancy on that day, doesn't make sense from a business perspective. The point of the Agile movement is that the old, micromanaged development model doesn't make good business sense either. The real focus of Agile is to once again put the customer's needs at front and center, where they belong.

The Agile Manifesto

Lightweight development methods such as Scrum had been in use since the mid-1990s but were known by few people other than the teams that were using them. Agile became a movement in February 2001, at a meeting at the Snowbird Resort in Utah. There, 17 software developers got

together and refined the underlying principles of lightweight methodology into the Agile Manifesto.

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

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

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

Source: http://agilemanifesto.org

The original drafters of the Manifesto wrote 12 Principles of Agile soon afterward. These principles grew out of the Manifesto and were written to provide more specific guidance to teams that wanted to use Agile for software development. The 12 Principles are listed in the Appendix of this book.

The goal of the Manifesto is to reintroduce empirical principles to software development. I say "reintroduce" because in the early days of coding, before software became a multibillion dollar industry, empirical development was the norm: that is, it was assumed that the realities of the actual project should guide the direction of the project. Blind obedience to the plan, even when the plan was wrong, came later, when mammoth budgets and gigantic teams seemed to require a book-length master plan laid out in advance.

A Look at Traditional Software Design Methodologies

Agile and other lightweight development models were created in reaction to heavyweight models, so it makes sense to compare the two approaches and use the contrasts to understand their differences. Heavyweight models are sometimes

called "predictive" or "plan-driven" models. They rely on predicting every possible variable before the project begins and writing a plan to address all of those variables.

Waterfall

The waterfall method is the most commonly used and best known predictive development model. The project moves from phase to phase (waterfall to waterfall) according to the plan. The current phase must be completed before the project can move on to the next phase. Customers are involved only in the requirements and deployment phases (the beginning and end), except in the rare instance where requirements change in the middle of the project.

- Create requirements and designs
- Develop the product
- Integrate the product with other products
- Test the product
- Fix problems
- Deployment

Intermediate Methodologies

Even before the Agile Manifesto was written, developers began to recognize the need for more flexibility in the development process, particularly on projects where the requirements couldn't be specifically known at the start of development. These methodologies can be considered forerunners of Agile.

- Iterative approach: allows for user input and feedback when the product is delivered and uses the feedback to make improvements
- Incremental approach: shortens the release cycle by delivering a working portion of the product in smaller pieces
- Spiral methodology: introduced in the late 1980s, this approach introduced prototyping as a way to get user feedback before the product's release
- Hybrid waterfall: phase-based methodology where you start the next phase as you're finishing the previous phase
- Rapid Application Development (RAD): like

the Spiral methodology, this approach also relied on prototypes and debuted in the mid 1990s

Agile methodologies

By 1995, Scrum and XP had already made their first appearances and attracted a following that would later produce the Agile Manifesto. Agile methods are characterized by their iterative and incremental qualities, relying on short bursts of activity, stakeholder testing and feedback, and incorporation of that feedback to constantly improve the product.

- Scrum: the most popular and well known of the Agile methodologies, and the focus of this book
- Extreme Programming (XP): another well known variant of Agile, with special emphasis on customer involvement and satisfaction by making customer request for new features part of the development process
- Kanban: a lean development process

pioneered by Toyota in the 1940s to support just in time (JIT) manufacturing

Chapter 2: Agile Up Close

The Agile Manifesto in Action

In Chapter 1, we learned the Agile principles as set out in the Manifesto. Now let's look at what they mean in real life situations.

Individuals and interactions over processes and tools

Agile recognizes that only people can create value, while processes and tools are useless if they don't help people do their jobs – or worse, stand in their way. In an Agile environment, individuals not only decide how to carry out the process, but they determine the process itself. At all times, the emphasis is on individual control, involvement, and motivation.

For example, when I worked in a traditional, plan-driven shop, we had to use a daily spreadsheet to document everything we did down to the last detail. This led to the familiar story of each programmer spending over two hours a day just recording how he had spent his time. If a problem arose, it got worse. I would write up an issue report, which would be passed up the chain of command to an invisible decision maker, who would pass his solution back down the ladder for implementation.

These things don't happen in an Agile shop. We verbally report on our daily activities in a 15-minute stand-up meeting, and that's it. The person with a problem is empowered to do whatever it takes to solve the problem. If he can't come up with a solution, a quick verbal discussion with one teammate is usually all that's needed. Verbal discussion takes far less time than writing things down, and a quick summary at the next stand-up meeting helps the rest of the team learn from the experience. As a result, I actually know more, not

less, about what my teammates are doing.

Working software over comprehensive documentation

In a plan-driven environment, specification documents tend to almost become ends in themselves. Agile principles dictate that the end point of any project is producing finished, working software – not a stack of reports. To this end, Agile encourages developers to keep their documentation "barely sufficient," meaning that it only communicates the minimum amount of information to fulfill the document's purpose. When the document expands beyond this minimalist approach, Agile adherents call it "gold plated." A gold-plated document has extra layers added to it that make it look prettier but serve no practical purpose.

For example, when I first started working on Agile projects, I was shocked to learn there were no weekly status reports. After being conditioned to think of status reports as a foolproof way to keep

management happy, I had to reorient my thinking and realize that if a report doesn't take me closer to working software, it's a waste of time. By comparison, the technical specification sheet is necessary even under Agile methodology because it contains information that everyone needs to know and is easier to communicate in writing than verbally.

Customer collaboration over contract negotiation

Plan-driven software development confines customer input to the beginning phases. Once the deal is inked, the customer is out of the picture until the product is ready for release. The problem with this approach is that very little is known about the project until work commences. For this reason, Agile emphasizes the constant involvement of the customer throughout the development process. Because the product is built in small blocks (called sprints), the development team can be extremely responsive to changes in the plan without backtracking or wasted effort. While Agile

doesn't dispense with contracts altogether, it allows customers and other stakeholders become partners in the ongoing development process.

Responding to change over following a plan

In plan-driven development models, the plan is static and change is something to be avoided at all costs. The farther the project progresses, the more expensive and disruptive change becomes. Agile, on the other hand, welcomes change and even cultivates it as an important part of the development process. Stakeholders and the development team communicate on a near-daily basis about the end product and its features. Responding to change allows the team to keep pace with rapid changes in the marketplace, as well as new information learned during the development process.

On my first Agile project, I was still stuck in the "customer as obstacle" mindset and feared that so much transparency would inevitably result in customers micromanaging the development

process. Agile prevents this by maximizing stakeholder involvement in the deliverable – the "what" of the product. The "how" of creating the product is left in the hands of individual team members, who are empowered to decide which tasks are necessary to get the job done. By the end of the project, I was a believer.

Benefits of the Agile Approach

The benefits of using Agile can be summed up in three categories.

- Better product: the iterative approach of incorporating user feedback into the project means higher customer satisfaction and better return on investment
- Fewer failed projects: incremental releases keep projects on track and help developers recognize failures before they start
- Better control of project scope: new features are only added in response to user input,

resulting in less product bloat and fewer features that are never used

Why Agile Might Not Work for You

Although Agile methodology has been producing amazing results since its inception, there are some instances where it might not be appropriate. Here are some things to evaluate before you embrace Agile as part of your process.

- Collocation or distribution: if your team is collocated in one office, your Agile adventure has a greater chance of success than if the team is working remotely in different cities or countries
- Team member experience: because Agile flattens the management hierarchy, inexperienced team members can be left behind during the development process
- Team size: problems with scalability are an

ongoing criticism of Agile methodology. These issues are being addressed with some success, but nonetheless a team of more than 12 individuals should give you pause before adopting Agile.

- Team commitment: all team members should have a strong commitment to working under the Agile method and contributing to the success of the project. There is no room for "that's not my job."
- Management attitude: Agile cannot succeed in a corporate culture where management is unable to let go of the need to manage every facet of the development process. It is vital to the success of the project that individuals on the development team are empowered to decide how to produce the deliverables specified by stakeholders.

Chapter 3: Introduction to Scrum

My Agile experiences have all been with Scrum. Not coincidentally, they all have been good. I must sound like a bit of a cheerleader, and I suppose I am. I hope that as you grow to understand its structure, you'll see why Scrum is the Agile method that has withstood the test of time.

Scrum is a variant of the Agile methodology and dates back to 1995, six years before the Agile Manifesto was written. The Agile method was largely inspired by Scrum and sought to broaden Scrum's techniques into a set of general principles. Agile is a mindset, while Scrum is a set of practical instructions. Scrum is the most popular variant of the Agile method, and it seems to be the most appropriate for small startups, as long as all of the players understand it and are strongly committed to it.

Scrum Team Roles

We'll start by getting acquainted with the players in the Scrum process. By the way, if you're wondering whether the word scrum as used in this book comes from the game of rugby, the answer is yes. If you're familiar with rugby, the short skirmishes and intense teamwork involved will suggest some obvious parallels with Agile methodology.

Product Owner

The product owner has an ownership stake in the product and therefore is strongly impacted by its return on his or her investment. For this reason, Scrum methodology dictates that the product owner has special authority in two areas: the product backlog and the priority (see below). Simply put, the product owner is the only person who can order work done or change the priority of work

done. This division of authority allows the other players to focus on producing deliverables without concerning themselves with the variables of ROI.

Team Member

Team members are responsible for the completion of deliverables (called "user stories" in Scrum lingo). Their role is highly collaborative in that they hold daily conversations with one another, as well as with the product owner and other stakeholders, such as customers or internal users. Each team member typically has a special area of expertise that he or she brings to the team. However, each specialist is expected to work outside of that specialty and contribute to the team effort, as well as collaborating freely at all times.

Team members also have their own sphere of authority, just as the product owner does. They have the sole authority to decide who does what, and to estimate scheduling for a deliverable. These decisions exclude the product owner, thus restraining management from imposing duties on

individual team members from above and overloading the team with impractical scheduling requirements. The team remains responsible for the deliverables, but their input into how those deliverables are produced is valued and accepted. In my experience, most failures of Agile methodology to produce good results can be traced to a corporate culture that cannot accept this division of authority – and realistically, some simply can't.

The original Scrum team size is strictly defined as 7 + or – 2, meaning it should be no smaller than 5 members and no larger than 9. More recently, teams have become less orthodox about following these numbers, and some are experimenting with larger teams. Some very small startups and even individual entrepreneurs have integrated Scrum principles in their work and in their daily lives, although individuals need to create alternate methods of collaboration.

Scrum Master

The master is a team member with one important extra area of specialty: hands-on expertise in the implementation of Scrum. He or she coaches other team members and helps them stay on track as they use Agile principles to get the job done.

The Scrum master is not a "boss" with any special authority. The roles of Scrum master and project manager have nothing to do with each other. The master is a respected peer who acts as a mentor and guide to make sure the Agile methodology is actually applied, and not just paid lip service.

The Structure of Scrum and Some Terminology

User Story
All activity in a Scrum environment begins with one or more user stories (sometimes just called stories). Stories define the product and are the reason for the project's existence. Each user story defines one or more specific deliverable items for

the development team.

A user story is simply a particular type of task, phrased in the following sentence:
As a [type of user], I want to [do something] so that [some value is created].
Users are always some type of stakeholder, either a product owner or a customer.

For example, a customer for a photo software program might tell the following user story:
As a customer, I want to adjust the image size and resolution of a photo for the web in one click to save time and frustration.

Product Backlog
Backlogs are lists in Scrum terminology. User stories define the product, so the product backlog is a list of user stories for the Scrum team to turn into deliverables. Each completed user story is called an *increment*, because Scrum is an incremental approach, and because completion of a user story incrementally increases the product's

value. Stories near the top of the product backlog have been split into smaller increments by the Scrum team so they can be tackled and finished in short iterations (called "sprints"), while stories at the bottom of the backlog tend to be larger and more broad as they await refinement.

The order in which the user stories appear in the backlog is called the *priority*. The product owner has control over the product backlog and the priority.

Acceptance Criteria
This is a set of pass/fail requirements developed jointly by the product owner and the development team. Each user story has its own acceptance criteria, which answers the question, "How do we know it's done?" Additionally, the entire product backlog has a set of acceptance criteria.

Artifacts
Artifacts are any tools used in the Scrum process, such as product backlog, user stories, and sprints.

In some Scrum settings I've worked in, the terms "product backlog" and "user stories" are used interchangeably.

The Sprint

Sprints are the heart of the Scrum methodology. A sprint is sometimes called a cycle, an iteration, or an increment, but the principle remains the same no matter what the name. A sprint is a short burst of activity involving the team, the product owner, and any other stakeholders that results in a piece of working software. The aim of a sprint is to produce a theoretically shippable product – it might be a very small product, and the product won't actually get shipped unless it makes business sense to do so, but nonetheless it addresses a user story and results in a deliverable.

Sprints are timeboxed, meaning they're constrained by time, not results. The longest sprints are usually capped at four weeks. Sprints as short as one week occasionally pop up, but two weeks seems to be the average time frame.

Sprint backlog

The sprint backlog is a list of tasks needed to complete a sprint. The team members have complete authority to define the tasks needed to complete one or more user stories assigned to them, and to decide how many user stories they can complete within the sprint's given time frame. Typically, by the time a user story reaches the top of the product backlog and is ready for a sprint, it will be quite small in scope, so the sprint will address multiple user stories at once.

Scrum terminology can be confusing regarding the word "task." Scrum defines user stories as a type of task, yet tasks are also the steps needed to complete the user story and produce a deliverable. I sorted out this concept by thinking of user stories as both "what" and "how." The product owner defines the "what" part of the user story in the beginning, when it's still very broad. As the story rises toward the top of the product backlog, the task is still "what," but it gets split into sub-tasks

and becomes more specific. By the end of the sprint, the "what" of the user story morphs into "how" as team members actually carry out the task it contains, refining and testing it as a deliverable. At the same time, they also perform other tasks to make the deliverable a reality, such as designing a good user interface, writing code, and producing documentation.

Because sprints are timeboxed, the sprint backlog ends when the sprint does. The acceptance criteria for a sprint is the completion of the sprint backlog within the given time frame.

Inspect and Adapt

This is a continuous improvement principle that allows the team and stakeholders to evaluate the results at the end of each sprint and make immediate improvements. Because Scrum is both iterative and incremental, inspect and adapt is a natural part of the development process. Inspection occurs after each increment is completed, and adaptations are added with each iteration.

Chapter 4: Scrum Up Close: The Sprint Cycle

Although in an ideal world, there would be no meetings, the highly collaborative nature of Agile means they can't be eliminated entirely. Scrum addresses this issue head on by clearly defining a purpose and a process for each meeting during the development process.

Scrum also uses the term *ceremony* in place of the word meeting. While it might help with motivating team members who are allergic to meetings (and I count myself among them), I find the term confusing so I've stuck with calling them what they are: meetings.

The sprint cycle consists of the following meetings

that take place before, during, and after each sprint:

- Sprint planning meeting
- Daily scrum
- Story time
- Sprint review
- Retrospective

I examine each of these in detail in the sections that follow.

Sprint Planning Meeting

As the name implies, this meeting takes place before the sprint. Its primary purpose is to develop user stories and agree on which ones will be delivered at the end of the sprint.

Out of all the Scrum meetings, this one bears the most similarities to traditional meetings, and extra care needs to be taken by the participants to keep it focused and on track. Participants should allow up

to 8 hours of planning time per sprint to accomplish the following:

- All participants agree on the length of the sprint, typically 2 weeks
- The product owner develops deliverables in the form of user stories, with the team's input
- The team examines the user stories and commits to one user story at a time until it feels it can't complete any more of them within the allotted sprint time span. The emphasis here is that the team, not the product owner or stakeholders, has control over the workload during the sprint.
- The product owner sets acceptance criteria for each user story. Although the owner has primary control over acceptance criteria, the collaborative nature of Scrum allows the team to have input. The result is that both the product owner and the team have the same picture of what the deliverables will look like.
- The team identifies the tasks needed to

complete the user stories by breaking each story down into a series of tasks. This is where the "what" of each user story is transformed into "how." The team has primary control over definition of tasks, but the product owner should be present to answer questions.

- The finished product at the end of the sprint planning meeting is the sprint backlog, consisting of a set of user stories and a list of tasks needed to accomplish each one. The sprint backlog is often drawn visually on a whiteboard or other tracking tool.

Daily Scrum

The daily scrum is sometimes referred to as the stand-up meeting. It takes place at the start of each day during the sprint, and its main characteristic is that it's kept short, no more than 15 minutes, so the team members can get to work producing a deliverable product. For this reason, the meeting is

held standing up.

The ceremony is quite simple: each team member simply recites a three-step summary:

- What I've done since our last daily scrum
- What I'll do before our next daily scrum
- Obstacles I'm encountering

It's a common misconception that the daily scrum is for solving problems. It's not. It merely makes the other team members aware that a problem might exist. The individual empowerment principles of Agile allow the individual who is having the problem to use his or her expertise to solve it. They get to make the judgment call of if and when to involve one or more teammates in a solution if they can't solve it themselves. At the very most, a decision might be made at the daily scrum that a certain teammate might be the best person to address the problem, based on his or her special expertise.

The purpose of this meeting is to apply Scrum's inspect and adapt principle on a daily basis. Small problems are learned from and addressed at the earliest possible phase, and the solution is incorporated into all future tasks during the sprint.

Story Time

While this meeting isn't "official Scrum policy," I've found it extremely helpful in the startup settings I've worked in. The purpose of story time is to go through the product backlog, examine each user story, and refine it so it's ready for an upcoming sprint.

Without story time, the user stories are placed in the product backlog when generated and remain untouched until the sprint planning meeting, where the team commits to them and assigns them tasks. This long period where they go unexamined is a lost opportunity to analyze and refine each component of the story, including:

- The action to be performed
- The value created as a result
- The acceptance criteria

Story time allows the product owner and team members to apply knowledge gained from recent iterations to each user story in the product backlog. It also creates time for story splitting. An important principle of Scrum is that stories with the highest priority should be shorter and more specific. Without story time, the Scrum methodology doesn't provide an opportunity to "right-size" user stories until just before they are placed into the sprint backlog. The aim of story time is to remove stories from the realm of the theoretical and make them practical in the real world.

To eliminate any confusion, I want to add that the purpose of story time is not to review stories in a currently active sprint. A sprint in progress should be considered untouchable unless extreme circumstances justify stopping the sprint. If all the players have done their jobs up to this point

(including the productive use of story time), stopping a sprint should be a extremely rare occurrence.

Sprint Review

This meeting takes place after the timebox for the sprint expires and the sprint is finished. This is showtime, where the product owner, the stakeholders (customers or internal users), and the team get together and the team demonstrates the deliverable. The deliverable consists of all completed stories from the sprint. The typical length of this meeting is 1-2 hours for a 2-week sprint.

Since the goal of each sprint is to produce working software, accolades are a standard part of the sprint review, but the review is far more than just a feel-good session. The true goal is to get feedback from the product owner and stakeholders to add to the inspect and adapt process. The deliverables

from the sprint will be subject to continuous improvement throughout future sprints based on the feedback from the sprint review. This feedback also guides the next sprint meeting that launches another sprint cycle as new user stories are brought up out of the product backlog.

Retrospective

After the sprint review, the team meets privately. The focus of this meeting isn't the deliverable itself. Rather, the purpose is to identify how the team's process for generating the deliverable can be improved. These meetings are kept short and to the point. Each team member identifies one or two task-related items that could be improved for the next sprint cycle. In keeping with the Agile principle of individual empowerment, implementation of process improvement is left up to the individual team member. There are no memos from on high prescribing solutions to problems encountered during the sprint with

expectations that everyone will follow them.

Chapter 5: Agile in Action

This chapter traces the product development process under the Agile framework from concept through completion. It relies heavily on Scrum and the sprint cycle, but contains additional material on the initial planning process. My intent is to show you how Scrum can be adapted in a variety of settings, to produce a variety of products – including but not limited to software development.

Product Visioning and Generating the Product Backlog

Although user stories are the heart of the Agile methodology, frequently they are conceived in bits and pieces, a little at a time, and appear as fragments instead of a whole. At some point early in the product development process, the product

owner should hammer out an overall vision for the product. In some instances, this might mean taking a set of user stories and synthesizing them into one single overarching user story. In other cases, the product vision might be conceived as a single user story from the beginning, to be split into smaller stories at the first sprint planning meeting. Either way, it's worthwhile at the beginning of a project to establish an objective for the product, asking questions like, "what are we trying to achieve?" and "what does the end result look like?" Further planning should only proceed when all parties involved agree on the product vision statement. If there are other stakeholders, such as active investors or partners, it's vital that they buy into the product vision, so the results aren't a surprise to them. If the product owner prefers to involve focus groups, this is the stage for consulting them.

With the vision statement in hand, the product owner generates a series of user stories to place in the project backlog, prioritizes them, and sets acceptance criteria for each one. This should be

done in advance of the first sprint planning meeting.

The Sprint Planning Meeting

This is a two-part process and works best if a full day is set aside for it. The first part is to be completed in the morning, with the second part in the afternoon following a lunch break. The product owner and all team members, including the scrum master, should attend. Other stakeholders, such as users or customers, may attend if they have something meaningful to contribute, but their role should be limited to advising in areas where they have expertise.

During the first half of the day, the product owner pulls a set of user stories with the highest priority out of the product backlog and presents them to the team. The number of stories presented will vary according to the length of the sprint, but it is

suggested that the number of stories presented should equal 150 percent of what the team reasonably feels it can handle within the sprint's time limits. This provides flexibility for deciding which stories to develop and which ones to leave in the backlog when planning the sprint.

As the product owner presents each story, the scrum master facilitates an intensive question and answer process by the team members. It is vital that the team understand the acceptance criteria for each story as deeply as the product owner does. By answering the question, "how do we know this story is finished?" the team develops a set of goals for the sprint.

After lunch, the product owner and the team reconvene and complete the second half of the sprint planning meeting, where the sprint backlog is generated. Here the team assumes the lead role in the meeting by assigning a series of tasks to each user story and estimating the time needed to complete each one. Large stories are split into

smaller ones to make task assignment easier. Story splitting clarifies large stories by making them more specific.

Time estimation is one of the greatest challenges of planning the sprint. Again, the team takes the lead by assigning "story points" to each user story. A story point is a relative measurement, meaning it is used to measure the size and scope of a story against other stories in the sprint backlog. Some teams allocate one story point per team member per day, but this isn't a standard measurement. Ultimately, though, a story with four story points should take four times as long as a story with one story point.

Finally, the team sets a target for how many story points it can complete within the given timebox for the sprint. Here it's helpful for a team to look at its past performance on similar projects for guidance. The scrum master can help rein in any temptation to overcommit and keep the sprint goals squarely within a realistic estimate of what the team can do

within the given time frame.

The final step of the sprint planning meeting is generating tasks and having each individual team member commit to his or her share of those tasks. Tasks are not assigned, although the scrum master may make suggestions based on knowledge of individual expertise. The power behind the Agile method is the commitment each team member makes to fulfill the goals of the sprint by stepping forward and taking ownership of a particular task. The sprint planning meeting ends when all tasks for all user stories in the sprint backlog have been committed to.

The Sprint

Although Agile teams are beginning to experiment with distributed teams and electronic communication, my experience suggests that the methodology works best if the teammates are not only in the same building, but in the same room.

Free exchange of information is absolutely vital during a sprint.

Each day of the sprint begins with the daily scrum, or stand-up meeting. The scrum master's role is paramount here in making sure everyone is on time and the meeting stays on track. Each member should aim for a happy medium between too much information and not enough as they tell the team:

- Tasks I've completed
- Tasks I'm going to complete next
- Obstacles I'm running into

When obstacles arise, the scrum master facilitates any help the team member might need. If the team member says he or she doesn't need help and just wants the obstacle noted, then that member's wishes are respected. One-on-one conversations about the project or problem solving should be carried out after the meeting so it doesn't waste the time of team members who don't need to be involved.

As the team makes progress day by day through the sprint, the scrum master notes completed tasks on a "burndown chart," which tracks the team's progress through the sprint backlog.

Inspect and Adapt

At the end of the sprint, the team presents a potentially releasable product to the product owner at the sprint review. Other stakeholders are frequently present at this meeting, such as customers and/or internal users. This is a chance for the team to shine while collecting valuable feedback to use in later iterations during the product development process.

After the team's internal retrospective meeting, where its internal process undergoes self-evaluation and improvement, another sprint planning meeting is held and the cycle repeats until the product vision is fulfilled. At this point, the

team's work is finished.

Appendix
The 12 Principles of Agile

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of

conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Source: http://agilemanifesto.org/principles.html