THE EXPERT'S VOICE® IN SOFTWARE DEVELOPMENT

Agile Software Development with HP Agile Manager

LEARN AGILE MANAGER TO IMPROVE YOUR SOFTWARE DEVELOPMENT PROCESS

Liran Tal

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

friendsof™

Apress®

Contents at a Glance

Introduction

Agile development practices have been widely adopted in a variety of organizations, yet only a few tools are available to help make the practical process of managing agile teams less painful and more successful. HP Agile Manager is a purpose-built SaaS-based agile planning tool that provides a simpler, smarter way to manage collaborative development. Agile Manager can work for small startups and midsize teams and can scale up for bigger organizations as a cost-effective and flexible tool to apply agile techniques to improve your software development process.

This book combines agile software development practices with HP's Agile Manager tool, when used as a SaaS product or an on-premise solution, and will explain how to master key functionalities of Agile Manager. It teaches you how to build and plan releases, create a product's backlog, and manage day-to-day R&D activity such as addressing user stories and defects and reviewing the team activity with the help of informative dashboards. While this book is not designed to teach agile software development *per se*, it starts out with a chapter that covers agile software development practices and specifically focuses on the Scrum agile methodology and its related concepts.

Who This Book Is For

This book assumes no previous knowledge of agile tools, or agile in general, so developers, quality engineers, development team leaders, and other stakeholders in an R&D team will find this book serves them well in learning and applying agile methodologies in their teams and in successfully building and adopting an agile development framework in their software projects. This book will also be of great help for teams already using an agile framework who want to further improve on their delivery with a more flexible agile tool such as Agile Manager.

Young team leaders and engineering managers will especially benefit from this book by learning and understanding how to adopt Scrum agile processes for successfully delivering results in their teams.

How This Book Is Structured

This book is designed to walk you through the software development stages ofanagile software methodology. It begins by laying the foundation for the agile philosophy and continues with signingup for and setting up HP's Agile Manager tool to manage a software project. It continues with chapters covering how to create the product's backlog, how to create themes and features, how to manage sprints, and how to streamline the R&D team's activities with user stories and defects. Finally, the book covers how to use dashboards and widgets to track and monitor real-time team activity and ends by reviewing online resources for further learning about Agile Manager's features, support, and the community around it.

CHAPTER 1

The Agile World

To begin your journey with managing agile teams and using Agile Manager for streamlining an agile workflow, you'll dive into the agile world by learning about the concepts, terminology, and the agile software development methodology.

In this chapter, you'll learn the basics of the agile philosophy and working principles to set the mood for the rest of your agile journey. We'll cover the following topics:

- Introducing the basic concepts and traits of the agile philosophy
- Reviewing agile terminology, including roles in software development
- Reviewing agile software models

Introducing Agile Software Development

The prominent software development methodology in the 20th century was the Waterfall process model, which employs a sequential design and development process that originated in the manufacturing and construction industry. While the Waterfall process is still used throughout the industry today in some software projects, the agile philosophy, introduced in 2001, challenges the norm of developing software projects by providing a less rigid, more flexible framework for developing software, where a shippable and deliverable product is achieved quicker through progressing research and development (R&D) iterations.

Origins of Waterfall Model

The basics of Waterfall model workflows are about setting up a pipeline process chain, which begins with a requirements phase and early planning step that may account for up to 30 percent of the time invested in the project. The rationale behind investing so much time in ironing out the requirements lies in the claim, per Rapid Development: Taming Wild Software Schedules (Microsoft Press, 1996), that a bug found in the early stages of requirements analysis is cheaper in terms of cost and R&D effort than the same bug discovered later in the process.

Note The Waterfall model is one of many software project models available for anyone to adopt, and modifications on the original Waterfall model include the Spiral model, the more current V-model, and the Modified Waterfall.

Software projects then continue with the rest of the phases of software modeling, building, and testing, and finally continue with the last phase of delivering the final product and maintenance, as shown in Figure 1-1.

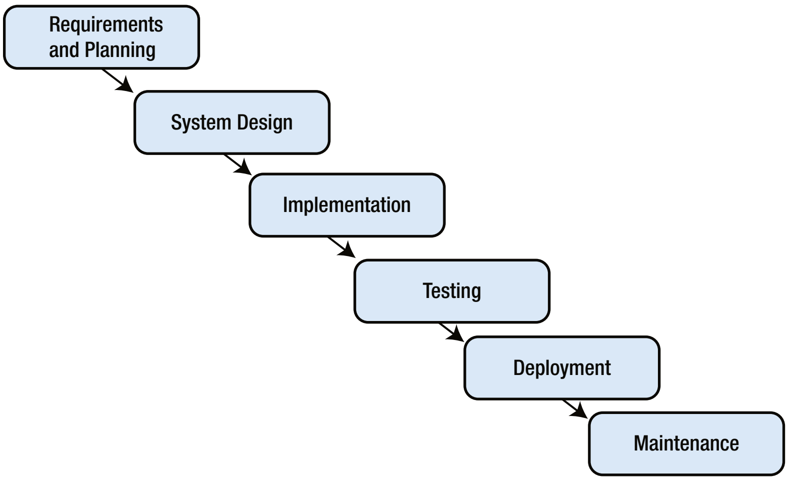


Figure 1-1. *Agile software development: Waterfall model*

Software projects developed with the Waterfall model provide a simpler approach to software development life cycles and provide solid requirements and general documentation assets throughout their initial phases, which enables new team members to easily join a project.

However, the following are the disadvantages of the Waterfall model:

- *Inflexible*: The Waterfall is unresponsive to changes. Once the specification requirements phase has locked in all the planned features, the project continues until completion.
- *Prone to bottlenecks and delays*: If an issue is detected at a late phase, such as in development or testing, this affects the entire process line because the phases are quite sequential, and it may introduce a significant delay time.
- *Late product delivery*: Only the completion of the final phases of the software project provide a working, deliverable product.

■ **Note** Today, you may still witness the Waterfall software development model employed for large projects or for corporations that haven't chosen to adopt a more agile workflow. Other reasons for organizations to employ a Waterfall process model is when technology, requirements, and product specification and definition are all stable, known, and considered constant. In other occasions, the Waterfall method may simply be a requirement of some governments and industries.

The Agile World of Software Development

Entering into the 21st century, in 2001 the first concepts of agile software development were introduced in the form of the Agile Manifesto (`http://agilemanifesto.org`). Some of the key values to the philosophy behind agile development processes are as follows:

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

These values mitigate and address concerns in previous, more rigid software development methodologies such as the original Waterfall model. It does not mean that processes, tools, documentation, or following a plan has no value but simply that the agile philosophy values these less when compared with other key values.

R&D **individuals and interactions**, collaboration, and continuous frontal team updates help to flush out any obstacles team members have and give room for ideas exchange and immediate feedback from teammates. It should be noted that agile methodologies don't reject tools or processes altogether but rather hint that fostering team members' interaction, as shown in Figure 1-2, is of higher value, and interactions shouldn't be replaced entirely by any tool or process.

Figure 1-2. *Individuals and interactions over processes and tools*

Having **working software**, even if it provides minimal value to the customer, is preferred over developing and documenting an entire software project whose life span can be relatively long and may turn out to be partially irrelevant when delivery is made. In other words, documenting every single part of the system by writing a top-notch requirements specification for every aspect in the product that will cover every feature to be developed is time-consuming and may often turn out to be a throwaway task because user requirements change; in addition, it's crucial to get users to interact with your software as soon as possible to be able to collect feedback and valuable input.

Documentation should not be disregarded or overlooked, though. It should exist, but it shouldn't be comprehensive in a way that stalls development or stalls the shipping of a minimal product to the customer to gain feedback. As shown in Figure 1-3, a product can be minimally functional and deliver customer value.

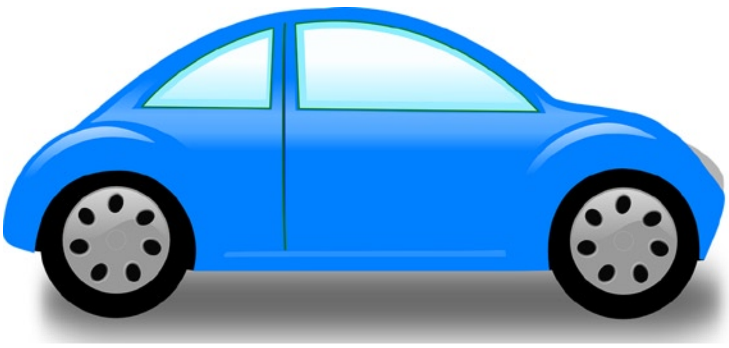


Figure 1-3. *Working software over comprehensive documentation*

Where contract negotiation limits the evolution of a software project because it ties down the agreed-upon features and developed content, agile promotes **collaboration with customers** to understand ongoing needs, reprioritization of features, and product vision for market alignment. Traditional software development models would lock in the customer to a feature set, and only then the project could continue with further design, development, and delivery. This meant that customers could evaluate the product only after everything had been developed and delivered and at a very late phase find out that the developers made bad decisions or that they were totally off with the market alignment and feature set. As shown in Figure 1-4, a customer's voice should be highly regarded and carefully examined, which can impact a product's feature set and vision altogether.

Figure 1-4. *Customer collaboration over contract negotiation*

All of these values are about being able to **respond to change** as software develops. Change should be embraced and prepared for with risk planning and modular, flexible software design. It is key to understand and accept that software evolves as users interact with it, and planned features may get thrown away or reprioritized. Agile software development acknowledges this ever-changing climate in software projects and offers solutions through processes to streamline these changes successfully.

Agile Flavors

The agile software development philosophy sets the mood for implementing the processes through the key values and principles. To support these values and build on the original agile principles, an emergence of agile methodologies and frameworks came to life to offer guidelines and lightweight processes to achieve agile development. These methodologies are also referred to as *flavors* of agile software development.

While there are many agile flavors, such as the more popular Scrum, Kanban, Extreme Programming (XP), and Feature-Driven Development (FDD), we won't cover all of them. Rather, we'll focus on Scrum, which is considered a common practice and which Agile Manager provides the toolbox for. Here are short reviews of selected agile methodologies:

- *Scrum*: In Scrum, products (or projects) are divided into iterative and time-boxed work units, referred to as *sprints*, which are typically two weeks long. A repetitive cycle composed of feature planning, user stories estimation, R&D work of developers and QA, and demos and retrospectives forms an iteration in which a potentially shippable product value can be met and delivered to the customer in minimal time.
- *Kanban*: Similar to Scrum, Kanban is an iterative process methodology that facilitates the delivery of product value through emphasis on optimizing the process flow. To make sure the process flow is adhered to, Kanban promotes the use of visualization of the flow's pipeline, often referred to as the *task board*, organized from left to right in flow of completion order, such as items that are in-progress development, done, in-testing items, done, and deployment. Items are put on the board and are moved in sequence from one column to another as they progress through the pipeline flow until completion.
- *Scrumban*: Merging both Scrum and Kanban, the Scrumban methodology desires to adopt the best elements from both and aims to integrate the task board workflow with Scrum concepts and foster the visualization of items. To that end, Scrumban promotes daily updates, review, and retrospective processes and combines them with Kanban work-in-progress (WIP) capacity limits and clear stages of execution.

■ **Note** Agile Manager also provides tools and workflows to effectively practice Kanban in your team, and even a Scrumban variation is quite common and well supported in the tool.

Practicing Scrum Agile Development

Scrum is a lightweight agile framework that is focused on making small teams work efficiently together to deliver a working product by employing methodologies and processes to streamline the work of the Scrum team for a successful software project.

The Scrum approach to software projects is through an iterative and incremental process, where team members continually work together to create a working product and continuously introspect the process for self-improvements. Figure 1-5 shows an example of such a process.

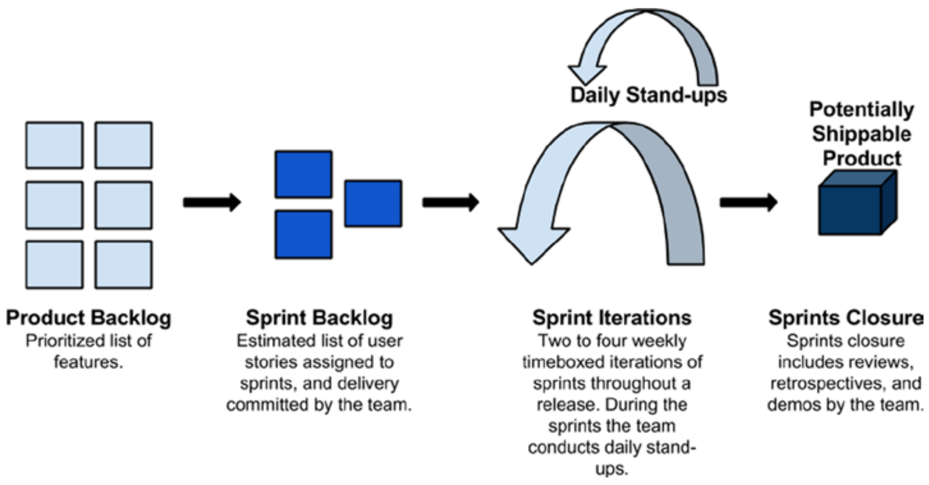

Daily Stand-ups
Potentially Shippable Product
Product Backlog
Prioritized list of features.
Sprint Backlog
Estimated list of user stories assigned to sprints, and delivery committed by the team.
Sprint Iterations
Two to four weekly timeboxed iterations of sprints throughout a release. During the sprints the team conducts daily stand-ups.
Sprints Closure
Sprints closure includes reviews, retrospectives, and demos by the team.

Figure 1-5. *Example of agile Scrum development life cycle*

■ **Note** While Scrum is well adopted and quite common in the technology industry, its processes can be customized and employed in other industries.

So you further understand the concepts of Scrum, we will lay the foundations of basic definitions in the world of an agile, and specifically Scrum, methodology.

Scrum Artifacts

The Scrum methodology identifies a few tangible deliverables that are the representations of specification requirements for the product or project.

The Product Backlog

The product backlog, as shown in Figure 1-6, is often referred to as the *general backlog* and is a prioritized list of requirements, or changes, that represent the desired feature set for the product. It is characterized as dynamic because requirements often change or are added.

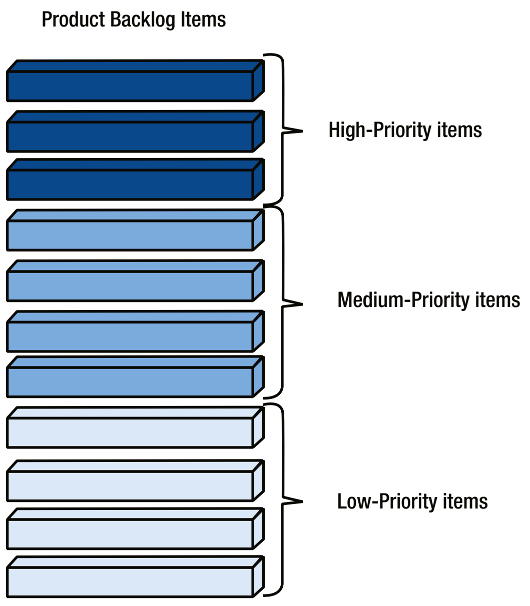


Figure 1-6. *The product backlog*

The content of the product backlog forms the basic idea pool for breaking the product backlog further into an organized set of deliverable value such as sprints and releases.

The following are backlog items that are referenced in the product backlog:

- *Themes*: Themes are general descriptive items for a focus area. For example, for a web application product, themes can be user management and messaging and notifications. Themes allow you to group similar interest area features.
- *Features*: Often referred to as *epics*, features describe a requirement in large and not so detailed specifics. Features can be thought of as a large set of tasks and items to develop (or large user stories) and are usually too large to complete in a single sprint. Features are broken down to smaller units called *user stories*, which are handled in the sprint backlog.

■ **Note** *Backlog grooming* is a common term for the practice of reorganizing the backlog items on the product backlog, such as reprioritizing them, removing them, creating new items, or correcting estimates. This activity is often referred to as the *backlog refinement meeting*.

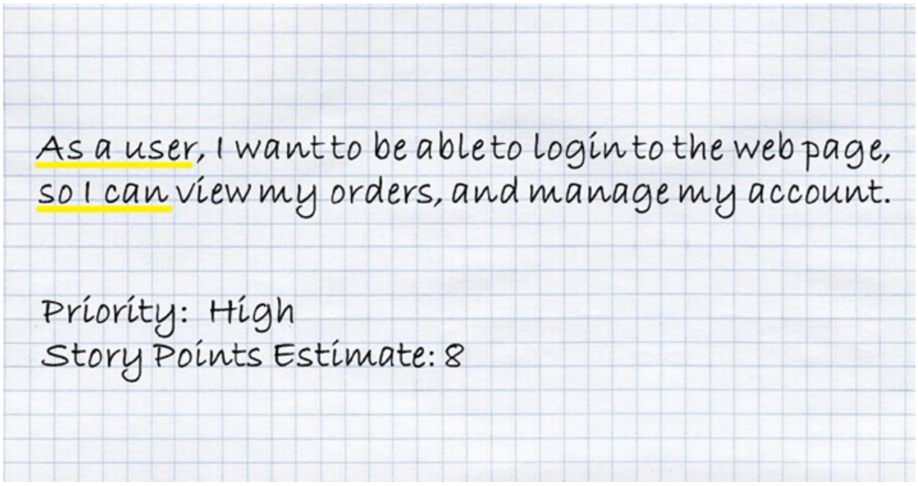
The Sprint Backlog

Sprints are time-boxed, iterative periods in which the team works together to deliver a set of product value items.

Similar to the general backlog, the sprint backlog is a set of defined requirements that is much more detailed and descriptive that the team has identified, discussed, and planned for completion in a given sprint. These items should be developed, tested, documented, and then integrated to provide a shippable and deliverable product.

The following are backlog items that are referenced in the sprint backlog:

- *User stories*: User stories are customer-focused descriptions for functionality that needs to be achieved in order to answer a set of product requirements. Figure 1-7 shows an example of a user story.



As a user, I want to be able to login to the web page, so I can view my orders, and manage my account.

Priority: High
Story Points Estimate: 8

Figure 1-7. *A typical user story*

- *Defects*: Defects and bugs are common backlog items that are tracked and logged by the QA team through sprint periods and are often used to indicate the quality of the product or developed functionality in a sprint.

■ **Note** Although it is common to use the term *bug* for describing problems in a feature or product, a bug and a defect aren't the same thing. A bug is the result of wrong or bad programming code, and a defect is a deviation from the requirements, a requirement that was not met, or a requirement that did not exist.

The following are general guidelines for user stories:

- *Simplicity*: User stories should be simple, short, concise, and often treated as small units of information that can fit on an A4 notecard.
- *Capture essential information*: User stories describe the who, what, and why of a requirement.
- *Estimated and prioritized*: User stories should be estimated using a measure like story points, which is a relative estimated effort, or using timed tasks. User stories should be small enough to complete within a single sprint; otherwise, they should be broken down into smaller user stories.

The Product Increment

A *product increment* is the set of product backlog items that have been completed in one or more sprints that provide value to the customer (see Figure 1-8). It is easy to understand product increments as releases, where each release, referred to as the *product increment* contributes to the final, more complete vision of the product and where real product value is provided in each release.

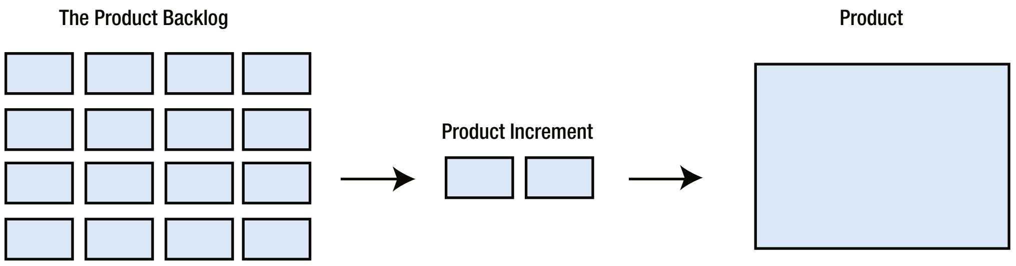


Figure 1-8. *The product increment*

Burndown Chart

The purpose of this chart is to represent the current state of the sprint in terms of the remaining work to be done. It is called a *burndown chart* because it should have a declining graph as the team walks toward the end of the sprint and work is decreasing toward completion (see Figure 1-9).

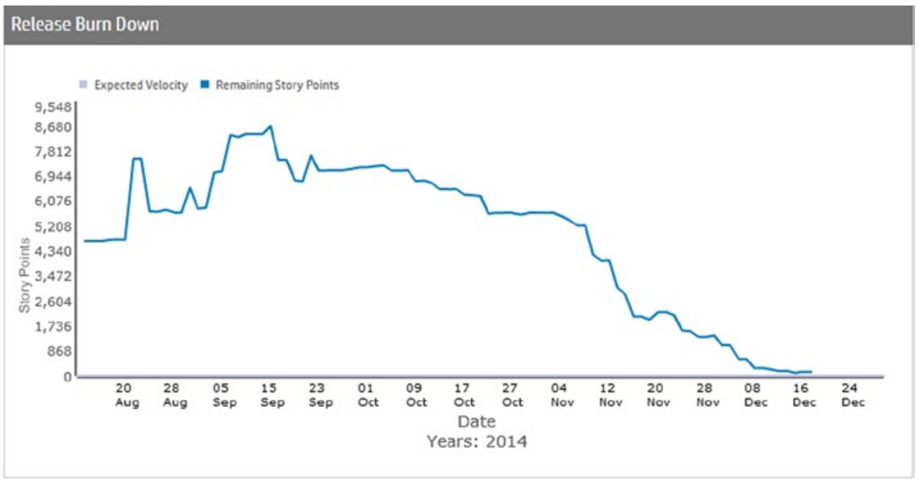


Figure 1-9. *The burndown chart*

Burndown charts show the remaining work on the y-axis and the sprint time length on the x-axis. On top of the actual remaining work items, the chart usually includes trend lines indicating the expected and actual remaining work and can be translated into the velocity of the team. That is, this is the speed in which backlog items are achieved throughout sprints.

Scrum Roles

The Scrum methodology defines a Scrum team that is composed of Scrum's own set of Scrum roles. These roles are not common titles like product manager or team leader.

- *Product owner*: The product owner manages the product backlog and ensures the backlog items are prioritized according to the product vision and are aligned with customer value and priority.
- *Scrum master*: The Scrum master is responsible for streamlining the Scrum team processes, activities, and practices to ensure that the Scrum methodology is followed.
- *The team*: The team includes the R&D team, developers, and QA team members working together to incrementally deliver shippable value products.

Scrum Sprints

In the Scrum methodology, software projects are facilitated through a predefined time window, usually several weeks long, of iterations that are referred to as *sprints*. Within these sprints, the team work together to achieve and complete parts of the project, through swift planning, R&D work, a closing retrospective, and demo sessions.

Figure 1-10 shows a simple and typical sprint iterative process; although the figure is incomplete, we'll fill in the details as we step you through the complete Scrum picture.

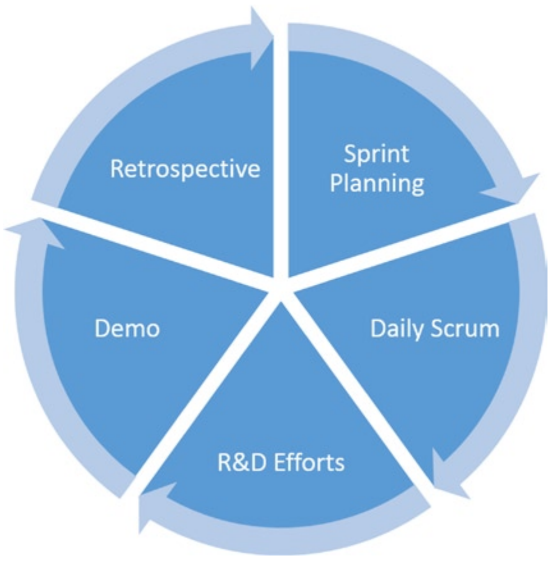


Figure 1-10. *Sprint iterative concept*

■ **Note** Demo sessions at the end of each sprint are important and allow the product owner to assess the fulfilment of the sprint backlog.