

AN INTRODUCTION TO AGILE SOFTWARE DEVELOPMENT

June 2007




TABLE OF CONTENTS

Executive summary.....	3
Agile vs. waterfall: practical differences in methodology	4
Two agile software development methodologies	6
XP	6
<i>The XP development process</i>	6
<i>XP rules and concepts</i>	7
SCRUM	8
<i>Scrum management</i>	8
<i>Scrum development</i>	8
<i>Scrum concepts</i>	9
How Serena supports agile software development.....	10
Conclusion.....	11

Executive summary

Agile software development (also called “agile”) isn’t a set of tools or a single methodology, but a philosophy put to paper in 2001 with an initial 17 signatories. Agile was a significant departure from the heavyweight document-driven software development methodologies—such as waterfall—in general use at the time. While the publication of the “Manifesto for Agile Software Development” didn’t start the move to agile methods, which had been going on for some time, it did signal industry acceptance of agile philosophy. A recent survey conducted by *Dr. Dobb’s Journal* shows 41 percent of development projects have now adopted agile methodology, and agile techniques are being used on 65 percent of such projects.¹



MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT²

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.

Figure 1. The agile software development manifesto

¹ Results from Scott Ambler’s March 2006 “Agile Adoption Rate Survey” posted at www.agilemodeling.com/surveys

² The Agile Manifesto site includes not only the original 17 framers, but also a large and continually growing list of supporters. Additionally, the site includes an overview of agile concepts and viewpoints of the original signatories. Visit <http://www.agilemanifesto.org/> to read about the manifesto.

Agile vs. waterfall: practical differences in methodology

The first software development methodologies were hardly methodologies at all, but a free-for-all as organizations struggled to profit from new computer-related technologies. As the industry learned more about developing software, certain techniques for managing and predicting the cost of software development projects came into use. The methodology that has dominated software development projects for decades is called “waterfall.” Winston Royce coined the term in 1970 to describe a serial method for managing software projects through the five stages shown in Figure 2.³

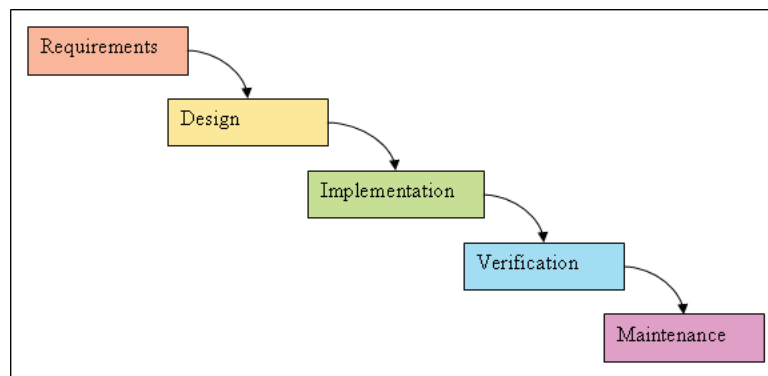


Figure 2. The waterfall process for software development

Adoption of waterfall has helped drive down the failure rate of software development projects, but even with rigorous project management and processes, a full 70 percent of software projects using this methodology fail to meet their objectives. To put this in perspective, waterfall software projects have less than half the success rate (66 percent) of going over Niagara Falls in a barrel.

Organizations tried to cut the failure rate by insisting on more detail in the requirements and design phases. This process of requiring extensive, even exhaustive, documentation culminated in 1988 with the publication of the Department of Defense Standard for software development, DOD-STD-2167A.⁴

³ Winston Royce, “Managing the Development of Large Software Systems,” IEEE WESCON, 1970.

⁴ 2167A was replaced by MIL-STD-498 in 1994. While this new standard did give software development organizations more leeway regarding methodologies, the required deliverables still correspond to stages within waterfall.

A manifesto for waterfall software development might look like Figure 3.

MANIFESTO FOR WATERFALL SOFTWARE DEVELOPMENT

Software development can be equated to any other engineering task. We believe software development projects can be effectively managed by:

- Understanding and **writing specifications** that define how the software will look and what it will do
- Performing in-depth **analysis and design** work before estimating development costs
- Ensuring software developers **follow the specifications**
- Testing the software after implementation** to make sure it works as specified, and
- Delivering the finished result** to the user

That is, if the specification is of sufficient detail, then the software will be written such that it will satisfy the customer, will be within budget, and will be delivered on time.

Figure 3. A possible waterfall manifesto

One of the most important differences between the agile and waterfall approaches is that waterfall features distinct phases with checkpoints and deliverables at each phase, while agile methods have iterations rather than phases. The output of each iteration is working code that can be used to evaluate and respond to changing and evolving user requirements.

Waterfall assumes that it is possible to have perfect understanding of the requirements from the start. But in software development, stakeholders often don't know what they want and can't articulate their requirements. With waterfall, development rarely delivers what the customer wants even if it is what the customer asked for.

Agile methodologies embrace iterations. Small teams work together with stakeholders to define quick prototypes, proof of concepts, or other visual means to describe the problem to be solved. The team defines the requirements for the iteration, develops the code, and defines and runs integrated test scripts, and the users verify the results. (See Figure 4.) Verification occurs much earlier in the development process than it would with waterfall, allowing stakeholders to fine-tune requirements while they're still relatively easy to change.

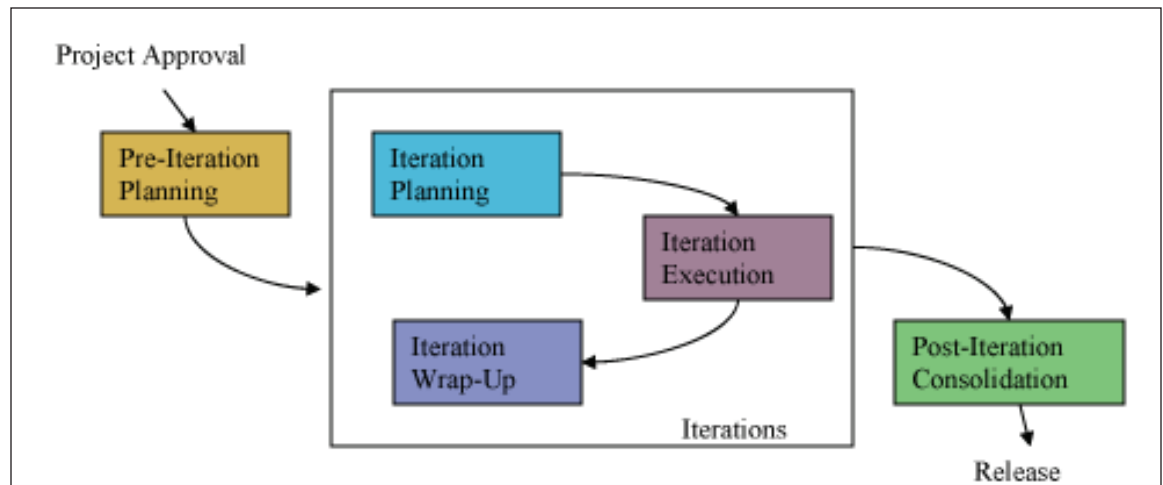


Figure 4. A generic agile development process features an initial planning stage, rapid repeats of the iteration stage, and some form of consolidation before release

Two agile software development methodologies

The most widely used methodologies based on the agile philosophy are XP and Scrum. These differ in particulars but share the iterative approach described above.

XP

XP stands for extreme programming. It concentrates on the development rather than managerial aspects of software projects. XP was designed so that organizations would be free to adopt all or part of the methodology.

XP development

XP projects start with a release planning phase, followed by several iterations, each of which concludes with user acceptance testing. When the product has enough features to satisfy users, the team terminates iteration and releases the software.

Users write “user stories” to describe the need the software should fulfill. User stories help the team to estimate the time and resources necessary to build the release and to define user acceptance tests. A user or a representative is part of the XP team, so he or she can add detail to requirements as the software is being built. This allows requirements to evolve as both users and developers define what the product will look like.

To create a release plan, the team breaks up the development tasks into iterations. The release plan defines each iteration plan, which drives the development for that iteration. At the end of an iteration, users perform acceptance tests against the user stories. If they find bugs, fixing the bugs becomes a step in the next iteration.

Iterative user acceptance testing, in theory, can result in release of the software. If users decide that enough user stories have been delivered, the team can choose to terminate the project before all of the originally planned user stories have been implemented.

Figure 5 shows a simplified version of XP. Full XP includes many steps in release planning, iteration, and acceptance testing that are not shown here. Visit www.extremeprogramming.org for a full description of XP.

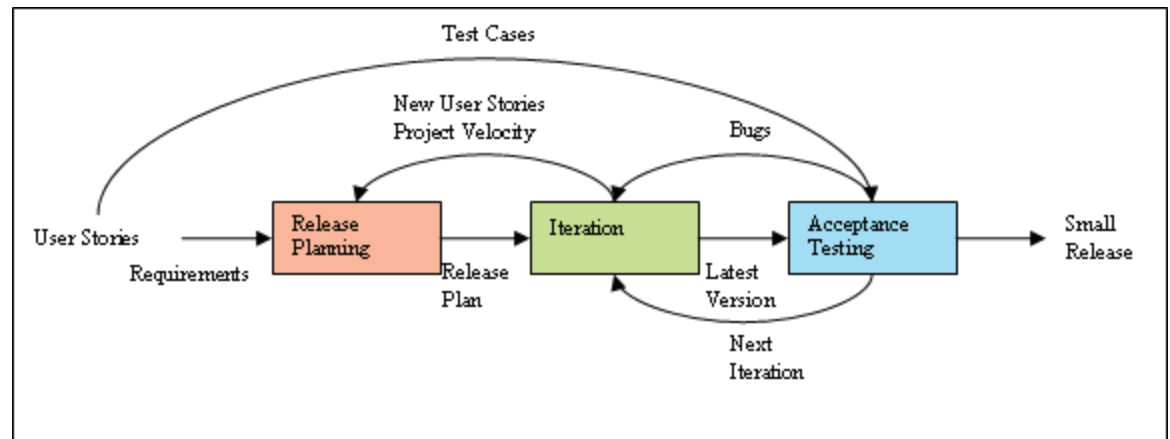


Figure 5. A simplified XP process

XP rules and concepts

The full list of XP rules and concepts is available at www.extremeprogramming.org/rules.html. Here are the most important concepts:

Integrate often. Development teams must integrate changes into the development baseline at least once a day. This concept is also called **continuous integration**.

Project velocity. Velocity is a measure of how much work is getting done on the project. This important metric drives release planning and schedule updates.

Pair programming. All code for a production release is created by two people working together at a single computer. XP proposes that two coders working together will satisfy user stories at the same rate as two coders working alone, but with much higher quality.

User story. A user story describes problems to be solved by the system being built. These stories must be written by the user and should be about three sentences long. User stories do not describe a solution, use technical language, or contain traditional requirements-speak, such as “shall” statements. Instead, a sample user story might go like this: Search for customers. The user tells the application to search for customers. The application asks the user to specify which customers. After the user specifies the search criteria, the application returns a list of customers meeting those criteria.

Because user stories are short and somewhat vague, XP will only work if the customer representative is on hand to review and approve user story implementations. This is one of the main objections to the XP methodology, but also one of its greatest strengths.

SCRUM

In rugby, 'scrum' (related to "scrimmage") is the term for a huddled mass of players engaged with each other to get a job done. In software development, the job is to put out a release. Scrum for software development came out of the rapid prototyping community because prototypers wanted a methodology that would support an environment in which the requirements were not only incomplete at the start, but also could change rapidly during development.⁵ Unlike XP, Scrum methodology includes both managerial and development processes.^{6,7}

Scrum management

At the center of each Scrum project is a backlog of work to be done. This backlog is populated during the planning phase of a release and defines the scope of the release (see Figure 6).

After the team completes the project scope and high-level designs, it divides the development process into a series of short iterations called sprints. Each sprint aims to implement a fixed number of backlog items (see Figure 6). Before each sprint, the team members identify the backlog items for the sprint. At the end of a sprint, the team reviews the sprint to articulate lessons learned and check progress.

During a sprint, the team has a daily meeting called a scrum. Each team member describes the work to be done that day, progress from the day before, and any blocks that must be cleared. To keep the meetings short, the scrum is supposed to be conducted with everyone in the same room—standing up for the whole meeting.

When enough of the backlog has been implemented so that the end users believe the release is worth putting into production, management closes development. The team then performs integration testing, training, and documentation as necessary for product release.

Scrum development

The Scrum development process concentrates on managing sprints. Before each sprint begins, the team plans the sprint, identifying the backlog items and assigning teams to these items. Teams develop, wrap, review, and adjust each of the backlog items (See Figure 6).

During development, the team determines the changes necessary to implement a backlog item. The team then writes the code, tests it, and documents the changes. During wrap, the team creates the executable necessary to demonstrate the changes. In review, the team demonstrates the new features, adds new backlog items, and assesses risk. Finally, the team consolidates data from the review to update the changes as necessary.

⁵ Scrum management methodology actually originated earlier in the manufacturing community and was adopted and adapted by software developers.

⁶ Some development organizations have combined Scrum managerial processes with XP development processes. The adoption rate is still rather low, but growing.

⁷ See www.scrumalliance.org for more information about Scrum.

Following each sprint, the entire team—including management, users, and other interested parties—demonstrates progress from the sprint and reviews the backlog progress. The team then reviews the remaining backlog and adds, removes, or reprioritizes items as necessary to account for new information and understanding gathered during the sprint.⁸

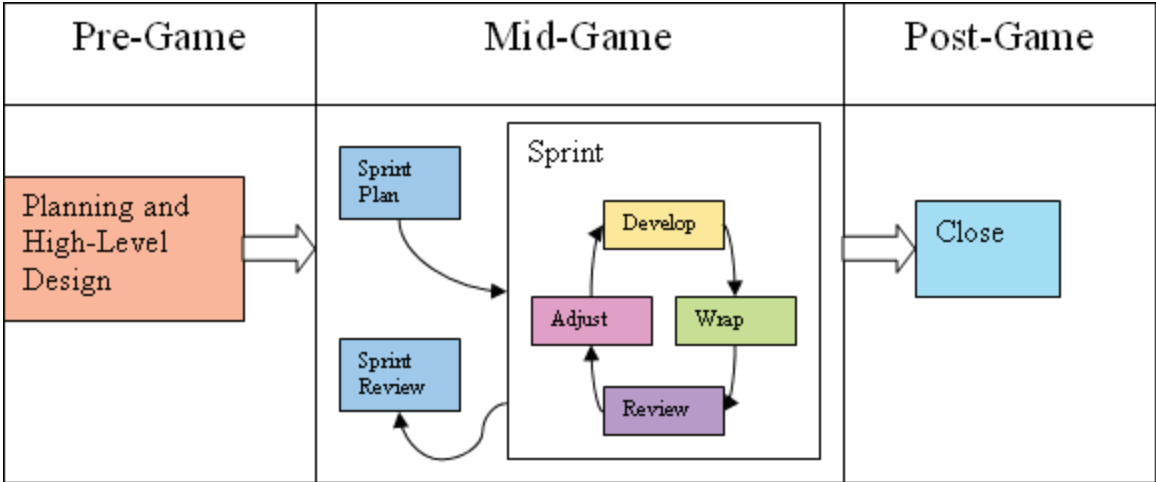


Figure 6: The Scrum process

Scrum concepts

For more detailed information on Scrum, refer to one of the many books on the topic, visit www.scrumalliance.org, or visit wikipedia. Here are a few of the most important concepts:

Burndown chart. This chart, updated every day, shows the work remaining within the sprint. The burndown chart is used both to track sprint progress and to decide when items must be removed from the sprint backlog and deferred to the next sprint.

Product backlog. Product backlog is the complete list of requirements—including bugs, enhancement requests, and usability and performance improvements—that are not currently in the product release.

ScrumMaster. The ScrumMaster is the person responsible for managing the Scrum project. Sometimes it refers to a person who has become certified as a ScrumMaster by taking ScrumMaster training.

Sprint backlog. Sprint backlog is the list of backlog items assigned to a sprint, but not yet completed. In common practice, no sprint backlog item should take more than two days to complete. The sprint backlog helps the team predict the level of effort required to complete a sprint.

⁸ Some Scrum process definitions include sprint planning within the sprint review. Others cite these as distinct meetings.

How Serena supports agile software development

The fast-paced development and cross-silo coordination necessary for a successful agile project require organizations to visualize the scope of the project and the project schedule, orchestrate the integration and testing process, and enforce adherence to agile processes.

Visualization helps development organizations to understand the project schedule and see the scope both of the entire project and of each iteration within the project. Serena Mariner[®] helps teams visualize the available resources to make sure they have the time, money, and personnel to complete the project. Additionally, Mariner helps teams organize iteration or sprint plans to manage and schedule software releases. Serena TeamTrack[®] also helps teams stay aware of the scope of an agile project by keeping track of user stories within XP, managing project and sprint backlogs within Scrum, or tracking tasks for other methodologies.

The high-speed development schedule of agile methodologies works best when coupled with automated processes that orchestrate some of the more mechanical development tasks. For example, within both XP and Scrum, development teams must integrate their code into the project baseline often. Each integration event involves complete builds of the project baseline as well as automated testing to validate the new code. Automation supports successful continuous integration policies. Depending on the size of the organization and the complexity of its development processes, either the Serena PVCS[®] Professional[™] suite or Serena Dimensions[®] CM can provide process automation such as continuous integration builds.

Agile development isn't a managerial free-for-all. It requires discipline and adherence to processes, even when those processes are not burdensome. For example, users must review and approve changes before they are merged into the baseline, developers must review each other's code, and code must undergo unit tests. These processes act as checks and balances to give greater freedom to development organizations within a light-handed enforcement framework. Serena TeamTrack can help the development organization to enforce these agile processes.

Conclusion

Agile software development stresses rapid iterations, small and frequent releases, and evolving requirements facilitated by direct user involvement in the development process. Serena's application lifecycle management tools provide a framework to visualize scope, orchestrate mundane and repetitive development tasks, and enforce process. Unlike agile-specific products offered by agile-only vendors, Serena products are methodology-neutral and can be applied equally well to agile as well as more traditional serial development processes, so they can support all the development activities within an enterprise.

ABOUT SERENA

Serena is the leader in Application Lifecycle Management for distributed and mainframe systems. More than 15,000 organizations around the world, including 96 of the Fortune 100, rely on Serena software to automate the application development process and effectively manage their IT portfolios. For more information on Serena software and services, visit: www.serena.com

CONTACT

Learn more about the enterprise-wide power of Serena products by visiting <http://www.serena.com> or contacting one of our sales representatives in your area.

Serena Worldwide Headquarters

Serena Software, Inc.
Corporate Offices
2755 Campus Drive
Third Floor
San Mateo, California 94403-2538
United States

800.457.3736 T
650.522.6699 F
info@serena.com

Serena European Headquarters

Serena Software Europe Ltd.
Abbey View Everard Close
St. Albans
Hertfordshire AL1 2PS
United Kingdom

+44 (0)800.328.0243 T
+44 (0)1727.869.804 F
ukinfo@serena.com

Serena Asia Pacific Headquarters

360 Orchard Road
#12-10
International Building
Singapore 238869

+65 6834.9880 T
+65 6836.3119 F
apinfo@serena.com

