

Agile Software Engineering Principles

Agile Approaches to Software Development

Nathan Hughes - 1809708

January 13, 2019

Abstract

A crisis in software development was present in the 1990s. In response, the Agile Manifesto was created and from it multiple methodologies were developed in order to answer the problems from which the field was suffering. In this paper, two of the major methodologies are described and analysed.

1 Introduction

Possibly the earliest mention of a management model for software development that resembles the waterfall model was by Herbert Benington in 1956 [1]. The model is mistakenly attributed to Winston Royce [2], although Royce recognised this early model in his 1970 paper as a primitive step towards a more complete model [3]. In it, Royce describes 7 steps of which the model consists.

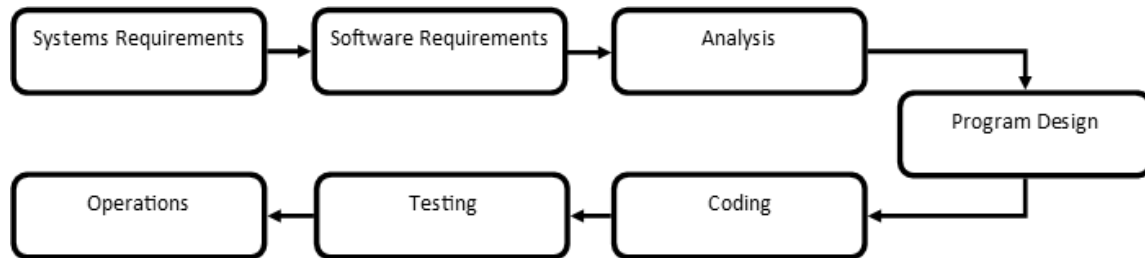


Figure 1: The fundamental structure of the waterfall model.

Contrary to the original model proposed by Benington, Royce recognised that there was a large risk present in following this model linearly and he proposed that it be approached iteratively and incrementally, repeating the cycle at least twice, and not necessarily in a linear fashion.

Despite this, the model produced many undesirable outcomes. According to research conducted by the Standish Group in 1994, 31.1% of projects were being prematurely cancelled, 52.7% of projects were costing at least 189% more than their intended budget and it was projected that American companies were going to spend \$81 billion on software projects that were to be ultimately cancelled [4]. The same study polled IT managers for opinions on why projects failed, and 'Incomplete Requirements' and 'Lack of User Involvement' were the most common responses. Companies that followed the waterfall model might not have communicated with their clients often after the initial

requirement gathering phase of their projects, so this response was to be expected.

Statistics such as this are representative of a crisis in the software development community at the time, and any solution would require a drastic change in the methods of management involved in the production of software. Some solutions were presented, such as the Scrum framework in the the Business Object Design and Implementation workshop of OOPSLA'95 [5] and Adaptive Software Development [6], but did not gain prevalence until some time later, with the introduction of the Agile Manifesto.

This report discusses aspects of Agile software development. Starting with an outline of the Manifesto, two of the most famous Agile methodologies, Extreme Programming and Scrum, are then discussed, along with offspring methodologies. Finally, a criticism of Agile is then offered.

2 The Agile Manifesto

The Agile Manifesto was collaboratively created in 2001 by representatives of multiple software development frameworks, such as Scrum and Adaptive Software Development, who sought a solution to the crisis of the 1990s [7]. The manifesto consists of 4 values and 12 principles, which are listed below [8, 9].

Values

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

Principles

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 time scale.
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 sponsor, 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 maximising the amount of work not done - is essential.
11. The best architectures, requirements, and designs emerge from self-organising teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Of particular note is the lack of any concrete framework offered by the manifesto. Rather, the crux is for methodologies to be created using these principles and values in order to successfully develop software.

In his book 'Agile Software Development Ecosystems', Highsmith suggests that the difference between waterfall and agile approaches to software development lie in two different assumptions that each make [10]. Traditional methodologies will assume that the customer is not able to fully understand the requirements of the software that they want when the project begins, and they assume that the customer is limited in their ability to detect future requirements. Agile approaches reject these assumptions, opting to learn with the customer continuously.

3 Extreme Programming

Extreme Programming was formulated by Kent Beck while working for the Chrysler Corporation [11]. It is described by Beck as both a methodology created for small to medium-sized teams working on products with requirements that change frequently, and places emphasis on collaboration between developers, managers and customers to achieve the final product [12]. Extreme Programming is employed by 1% of respondents to the State of Agile 2018 survey [13].

3.1 Outline of Methodology

Extreme Programming is detailed extensively in Beck's book [12]. The fundamentals of Extreme Programming as described by Beck include:

1. Test first development: unit tests are to be created before the code proper is written,
2. Continuous integration: the whole program is to be integrated and tested several times a day,
3. Pair programming: all coding should be conducted in pairs sharing one screen,
4. Initial simple design: the first iteration of the product should be simple and built upon, removing unneeded complexity as this process continues.
5. Minimum value product: the minimal system should be launched quickly and should be grown according to value gained.
6. Incremental: Requirements of the product are to be obtained and defined incrementally as the product is created.

Extreme Programming plans the process through the *Planning Game*, which occurs once per iteration, which is usually a week long. This is divided into two parts; *Release Planning* determines the requirements for release in the short-term and when they should be released, and *Iteration Planning* determines the tasks the developers are to accomplish.

3.2 Criticism of Extreme Programming

Critics of Extreme Programming have pointed to multiple potential flaws in the methodology, many of which are discussed by Carroll and Morris in their book 'Agile Project Management in Easy Steps, 2nd Edition' [14]. The design of the product is designed incrementally from the minimum viable product, meaning that more unnecessary redesigning is needed. Adding to this, since the ethos of Extreme Programming is continuous and incremental definition of product requirements, the customer is required to meet with the development team frequently, adding to time and money costs. The Extreme Programming methodology is immutable and unable to be implemented partially. As Beck explains, the practices are chosen not because they are flawless, but because each practice has strengths that compensates for the weaknesses of others [12]. As such, not including all practices is likely to emphasise the flaws of the rest, possibly leading to catastrophic project failure.

A case study on the topic of Extreme Programming was conducted by Layman, Williams and Cunningham at Sabre Airline Solutions [15]. The same product was released by the same team, one without the use of Extreme Programming, the other having used the methodology for approximately two years. The study found that the team had increased its productivity by 50%, with a 65% in pre-release quality and 35% increase in post-release quality. These statistics suggest that the methodology is able to improve the quality of work produced by teams that are experienced in it's implementation. The team in the study was small-scale and localised, and had managerial and organisational support while working on the projects, so care should be taken in extrapolating these results to larger teams with no existing support structure.

Specifically on the topic of paired programming, a contentious issue with critics, a study by Hulkko and Abrahamsson failed to conclude that paired programming consistently increases productivity of developers when compared to solo programming [16].

4 Scrum

Scrum is probably the most popular agile methodology currently being implemented, with 56% of respondents to the State of Agile 2018 survey [13]. The name 'Scrum' was chosen as a comparison to a rugby team, with coordinated strategies employed by high-performing, cross-functional development teams to achieve a common goal [17].

4.1 Outline of Methodology

The creators of the Scrum methodology have written a guide to the implementation of Scrum, which is summarised here [18]. As with most methodologies, Scrum begins with the initial phase of requirements gathering. These are then refined into user stories which are kept in the *Product Backlog*, and it is the responsibility of the *product owner* to manage and maintain this Backlog. It is the goal of the development team to eventually complete the Product Backlog.

The central structure of the Scrum is the *Sprint*. This is a set period of time, usually less than a

month long, during which a subset of the Product Backlog, called the *Sprint Backlog* is chosen to complete by the Scrum Team. The Sprint is commenced by choosing the Sprint Backlog. Every following day begins with the *daily Scrum*. During this 15 minute period, developers will discuss what they accomplished the previous day, what they plan on doing today and if there exists any impediments that prevent them from finishing the Sprint. The role of the *Scrum Master* is to ensure that the scope and goals of the Sprint are understood by everyone involved in the Scrum, removing the impediments that the development team have come across, facilitating the events such as Daily Scrum and helping the development team self-organise.

After the Sprint is over, a *Sprint Review* is conducted. This is a meeting attended by the Scrum Team and any stakeholders that the product owner has invited. They discuss what has been achieved during the sprint and how successful the Sprint was. This can also include demonstrations of the work accomplished during the Sprint. If the requirements of the product have changed, this will also be discussed here, producing a revised Product Backlog ready for the next Sprint.

Following this event is the *Sprint Retrospective*. This is an opportunity for the Scrum Team to examine the previous Sprint, identifying what went well and what could potentially be improved. A plan is created for implementing these improvements. This process is iterated until the project is completed.

4.2 Criticism of Scrum

Scrum is designed to be flexible for the full lifetime of the project. Since the project requirements are able to be modified consistently throughout the process, adaptations responding to new environmental variables, such as new user requests and emerging markets of potential users, is achievable which waterfall methods are less capable of doing. This is consistent with the fourth value of the Agile Manifesto. This is made easier by having only one input interface, the Product Backlog. As Ionel notes in his critical analysis of Scrum, object-orientated technologies make the implementation of Scrum easier than procedural technologies, since it can be employed selectively when interfaces between objects are simple enough [19]. For more complex interfaces, other methodologies should at least be considered.

One clear disadvantage of Scrum is scope creep, which can be a result of a client with wide vision. On the other hand, clients with little or no clear vision of the product can create a product that is different from expected [20]. Additionally, while the client is allowed to change the definition of the product, the period of time in which they can affect the backlog is periodic, at the end of each sprint, and so the rate of change might be as long as a month.

An important variable that is to be considered before adopting Scrum is the size of the team. Since independence is emphasised by the creators of Scrum, smaller teams are ideal since they are more capable of conducting independent work efficiently than larger teams [21]. Frameworks such as LeSS (Large-Scale Scrum) have been developed to accommodate larger teams. This methodology offers more guidelines for the larger team to follow, with the goal of 'downscaling' the structure of the team [22].

In a 2012 case study conducted by Hajjdiab, Taleb and Ali, an organisation with multiple teams and projects was tasked with adopting the Scrum methodology. This process ultimately failed due to many factors, including the absence of an Agile coach, team members impatient to adopt the methodology as quickly as possible and the absence of a pilot project [23]. The team experienced difficulty in implementing the methodology since it differed so greatly from the waterfall method that they had previously employed.

4.3 Usefulness in Industry

A 2008 study by Salo and Abrahamsson gathered survey responses from 35 projects and found that those that practised the Scrum methodology reported more positive experiences compared to those that had no experience [24]. Although the sample size of this study was too small to make any empirical conclusion, it is certainly an indicator that the methodology is of use to development teams.

Multiple adaptations of Scrum have been developed, using parts and modifications of the methodology to better fit to the organisation using it. Hron and Obwegeser conducted a literature review of academic publications into these modifications and found that some of the more common modifications include:

1. combinations of Scrum and other methodologies such as CMMI, Lean or XP,
2. The introduction of extra artefacts or roles to deal with more specific tasks such as the creation of milestones, technical architecture definition writing and communicator of product vision,
3. multiplicity of roles, artefacts and processes, such as separate backlogs for developers, managers and testers,
4. Introduction of tools to facilitate the Scrum process [25].

One of the most popular of these modifications is known as Scrumban, an amalgamation of Scrum and Kanban, with an 8% adoption rate by surveyed projects in the State of Agile 2018 survey [13]. In his collection of essays on the topic, Ladas discusses the fixed-length iterations present in Scrum and other Agile methodologies, compares them to the Just-In-Time methodology in Kanban and advocates for the adoption of JIT over iteration, but retaining the good practices of Scrum [26].

5 Criticism of Agile

Referring back to the Standish Group 1994 report, Agile principles don't sufficiently meet some of the concerns of executive managers. Unrealistic expectations, lack of executive support and lack of IT management are examples of factors that impaired projects at the time that Agile principles are not capable of addressing [4]. In addition to this, in their 2012 article, Kumar and Bhatia argue that a shortcoming of Agile manifests itself in the lack of attention given to architectural and design issues [27].

Despite this, the Standish Group 2015 Report gathered that, with wider implementation of Agile methodologies, 29% of IT projects were successfully under budget and on time [28], compared to the 16.2% success rate in 1994. It is therefore hard to argue that Agile has not had a positive impact on productivity of development teams.

Many of the criticisms mentioned here were discussed in the 2011 Agile conference in Snowbird, Utah. In addition, there was a concern that the most important people involved in the dispersion of Agile as a methodology had financial interest in its success, and failures of Agile were being purposely censored [29]. To add to this, there was concern that the context in which methodologies succeeded or failed were not being communicated properly, further damaging the movement.

6 Conclusion

Agile as a movement has gained significant traction since the creation of the Manifesto in 2001, with over 52% of respondents to the 2018 State of Agile survey claiming that more than half of the teams in their organisations are using Agile methodologies to organise themselves [13]. The difference between Agile methodologies can be stark; this difference can be seen in by examining only the two presented in this report. This reflects the loose definitions provided by the Agile Manifesto.

As was previously discussed, adoption of Agile methodologies has seen an increase in project success and overall team satisfaction. While some methodologies have seen much more widespread adoption than others, the realisation of said methodologies differs between organisations according to need [25].

Agile as a guiding principle does have its drawbacks, however, and is certainly not a perfect solution to all of the problems that development teams are likely to face while working on a project.

References

- [1] H. D. Benington, “Symposium on advanced programming methods for digital computers,” ser. ONR symposium report, ACR-15. Office of Naval Research, Dept. of the Navy, June 1956.
- [2] B. W. Boehm, “Software process management: Lessons learned from history,” in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE ’87. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987, pp. 296–298.
- [3] W. W. Royce, “Managing the development of large software systems: concepts and techniques,” *Proc. IEEE WESTCON, Los Angeles*, pp. 1–9, August 1970, reprinted in *Proceedings of the Ninth International Conference on Software Engineering*, March 1987, pp. 328–338. [Online]. Available: <https://web.archive.org/web/20040330021615/http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>
- [4] J. Johnson and H. Mulder, “The standish group report 1994,” January 1994.
- [5] K. Schwaber, “Scrum development process,” in *Business Object Design and Implementation: OOPSLA ’95 workshop proceedings 16 October, 1995, Austin, Texas*, J. Sutherland and P. Patel, Eds. Springer, 1997, pp. 117–135.
- [6] J. A. Highsmith III, “Adaptive software development: A collaborative approach to managing complex systems,” January 2000.
- [7] K. Beck, M. Beedle, A. van Bennekum *et al.* History: The agile manifesto. [Online]. Available: <https://agilemanifesto.org/history.html>
- [8] —. Manifesto for agile software development. [Online]. Available: <https://agilemanifesto.org/>
- [9] —. Principles behind the agile manifesto. [Online]. Available: <https://agilemanifesto.org/principles.html>
- [10] J. A. Highsmith III, *Agile Software Development Ecosystems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [11] L. Copeland, “How to, extreme programming,” *ComputerWorld*, December 2001.
- [12] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston, Massachusetts: Addison-Wesley Professional Publishing, 2003.
- [13] VersionOne, “The state of agile 12th edition,” CollabNet, Tech. Rep., 2018, accessed: 12-01-2019. [Online]. Available: <https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report>
- [14] J. Carroll and D. Morris, *Agile Project Management in easy steps, 2nd edition*. Warwickshire, UK: In Easy Steps Limited, 2015.
- [15] L. Layman, L. Williams, and L. Cunningham, “Exploring extreme programming in context: an industrial case study,” in *Agile Development Conference*, June 2004, pp. 32–41.

- [16] H. Hulkko and P. Abrahamsson, “A multiple case study on the impact of pair programming on product quality,” in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE ’05. New York, NY, USA: ACM, 2005, pp. 495–504. [Online]. Available: <http://doi.acm.org/10.1145/1062455.1062545>
- [17] (2014) What is scrum? Accessed: 12-01-2019. [Online]. Available: https://www.scrumalliance.org/ScrumRedesignDEVSite/media/ScrumAllianceMedia/Public%20Relations_2/What-is-Scrum-Backgrounder-2014.pdf
- [18] K. Schwaber and J. Sutherland. (2017, November) The scrum guide. Accessed: 12-01-2019. [Online]. Available: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>
- [19] N. Ionel. (2008) Critical analysys of the scrum prject management methodology. The Academy of Economic Studies Bucharest. Accessed: 13-01-2019. [Online]. Available: <http://steconomiceuoradea.ro/anale/volume/2008/v4-management-marketing/077.pdf>
- [20] J. Highsmith and A. Cockburn, “The business of innovation,” *IEEE Computer*, pp. 120–122, 2001.
- [21] R. L and J. N.S., “The scrum software development process for small teams,” *IEEE Software*, 2000.
- [22] C. Larman and B. Vodde, *Large-Scale Scrum: More with LeSS*. Boston, Massachusetts: Addison-Wesley Professional Publishing.
- [23] H. Hajjdiab, A. Shaima Taleb, and J. Ali, “An industrial case study for scrum adoption,” *JSW*, vol. 7, pp. 237–242, January 2012.
- [24] O. Salo and P. Abrahamsson, “Agile methods in European embedded software development organisations: A survey on the actual use and usefulness of Extreme Programming and Scrum,” *Software, IET*, vol. 2, pp. 58–64, March 2008.
- [25] M. Hron and N. Obwegeser, “Scrum in practice: an overview of scrum adaptations,” in *Proceedings of the 51st Hawaii International Conference on System Sciences*, January 2018.
- [26] C. Ladas, *Scrumban*. Seattle, WA, USA: Modus Cooperandi Press, 2008.
- [27] G. Kumar and P. Bhatia, “Impact of agile methodology on software development process,” *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, vol. 2, pp. 2249–6343, August 2012.
- [28] S. Group, “The standish group report 2015,” 2015.
- [29] P. Krutchen, “Agile’s teenage crisis?” *InfoQ.com*, June 2011, accessed: 13-01-2019. [Online]. Available: <https://www.infoq.com/articles/agile-teenage-crisis>