

ACCA7001 AGILE SOFTWARE ENGINEERING PRINCIPLES

Agile approaches to software development

Abstract

This document discusses the agile approach to software development and compares and contrasts it against the traditional approach, including explanations of the different methodologies they both encapsulate.

Edward Prasad
(1809704)

Table of Contents

Table of Contents.....	1
Introduction	2
Discussion.....	3
Incremental vs Iterative	3
Agile Overview	4
Scrum – Description and Case Study	5
Case Study.....	7
Conclusion.....	7
References	8

Introduction

This document will discuss the agile approach to software development and compare and contrast it against the traditional approach, including discussing the different methodologies they both encapsulate.

To begin with the software development lifecycle must be defined as both the traditional and agile approaches utilise it. The software development lifecycle (SDLC) is defined by a number of processes that developers follow to create high quality software and it allows all parties involved, such as developers and stakeholders, to know how far along the software is in the development process [1]. The processes of the SDLC are as follows: *(i)* requirements gathering, *(ii)* analysis, *(iii)* design, *(iv)* implementation, *(v)* testing and *(vi)* deployment [2].

(i) Requirements gathering is a very important stage of the development as it is the stage where the requirements and features of the software are defined and recorded. Requirements can be anything that the software will need, from frontend user interaction, to computational features, to backend infrastructure. The requirements can be gathered in several ways including questionnaires and unstructured interviews with end users but also in more formal circumstances such as in a meeting between the client and developing company. Often the requirements are detailed and recorded in a formal document known as the software requirements specification (SRS) [2]. The SRS acts as a legally binding contract between the client and the developing company while detailing the features of the software to be developed, including interfaces and infrastructure, and will detail any other business requirements needed. The requirements specified in the SRS are used in both the analysis and design phases of the SDLC [3]. *(ii)* The analysis section of the SDLC is where the developing company will: work through each of the requirements and expand upon them if needed; categorise and delegate similar requirements to certain teams or team members and organise the requirements by priority [2]. *(iii)* The design section is where the development team breakdown the requirements and develop solutions to the problems they present [1]. Multiple designs for both individual features and the architecture of the system are created and presented to stakeholders and senior members of the team who determine which designs will be used [1] [2]. *(iv)* The implementation section is where the features of the system are written and developed in the programming language specified during the requirements gathering and analysis stages. The designs created during the design phase are used to structure the features created during implementation. *(v)* Testing is the stage where the software created by the developers is passed to a team of testers who run the code against tests to ensure that the software behaves as intended and meets the defined requirements [2]. The tests are commonly written after or during the implementation process but with test first and test driven development, they can be written before the developers create any functional code. *(vi)* The first stage of deployment is allowing the client to interact with the system once it is completed. If there are any requirements that have not been met, or if the client wants extra features added, the whole process starts over again. Once the software has been signed off by the client it is deployed so that the final end users can use the system.

The traditional approach to software development was an incremental process that heavily followed the SDLC such as the waterfall methodology. Another example of a traditional approach that differs slightly, in that it is an iterative process, is the spiral model. The spiral model still follows the SDLC however iterates, and builds upon, previous versions to constantly improve the requirements and design. The agile approach to software development is both iterative and incremental, in that, there are defined stages throughout the development process which encapsulate more than one single process.

Traditional and agile methodologies and the difference between iterative and incremental development are discussed in greater detail below.

Discussion

Incremental vs Iterative

One of the most common methodologies of traditional software development is the waterfall model. The waterfall model has defined stages that follow the SDLC which are: analysis, design, implementation, testing and maintenance. This structure allows requirements to be defined in great detail early on in the development process and allows for the designs to be completely described in full before any code is written. All the implementation is completed in a single stage before testing and then the system is maintained after deployment [4]. Once the system is deployed, the client can use the system fully but if they want to add, or make changes to, requirements they must start a new process from the beginning of the methodology, which can become costly for the client. The waterfall model allows for the development process to easily flow from one section into the next but reversing through the stages can become difficult.

The waterfall model is an incremental methodology because each stage must be completed before the next can begin. Benefits of incremental design are that it ensures everything in a stage is completed before moving away and that all stages are clearly defined and understood by all parties. A problem with incremental design is that changing requirements becomes difficult, if not impossible, when the design and/or implementation have already begun as each stage needs the details of the last [4].

Iterative design is when a system is implemented and deployed to the client in multiple stages. After each deployment the client can add or change requirements to update the system. Each cycle of an iterative process delivers a working version of the system which eradicates errors and improves the features of the previous [5]. A big difference between iterative and incremental design is that there is a much shorter time between requirements gathering and implementation, in addition the client is more involved as they can see the system at the end of each cycle. Requirements do not need to be as strict or as in depth in iterative as the requirements are refined during each cycle [5]. This increase in requirement refinement can cause an issue called feature creep, which is when a client adds more features to a system than were initially stated [6]. As there is less initial documentation detailing requirements there are more chances for a client to add features that may not have been mentioned initially. A small amount of feature creep can be beneficial as there could be features that were not initially stated but are needed for the system to behave as intended. However, too much feature creep can mean initial features do not get implemented as the developers are spending too much time creating the additional features [6].

The spiral model (created by B. Boehm, 1988 [7]) is an iterative model where the development is split into four sections that are constantly repeated until the client is sufficiently happy with the deployed system. The four sections are: determining objectives, risk assessment, development and validation and planning, as shown in figure 1.

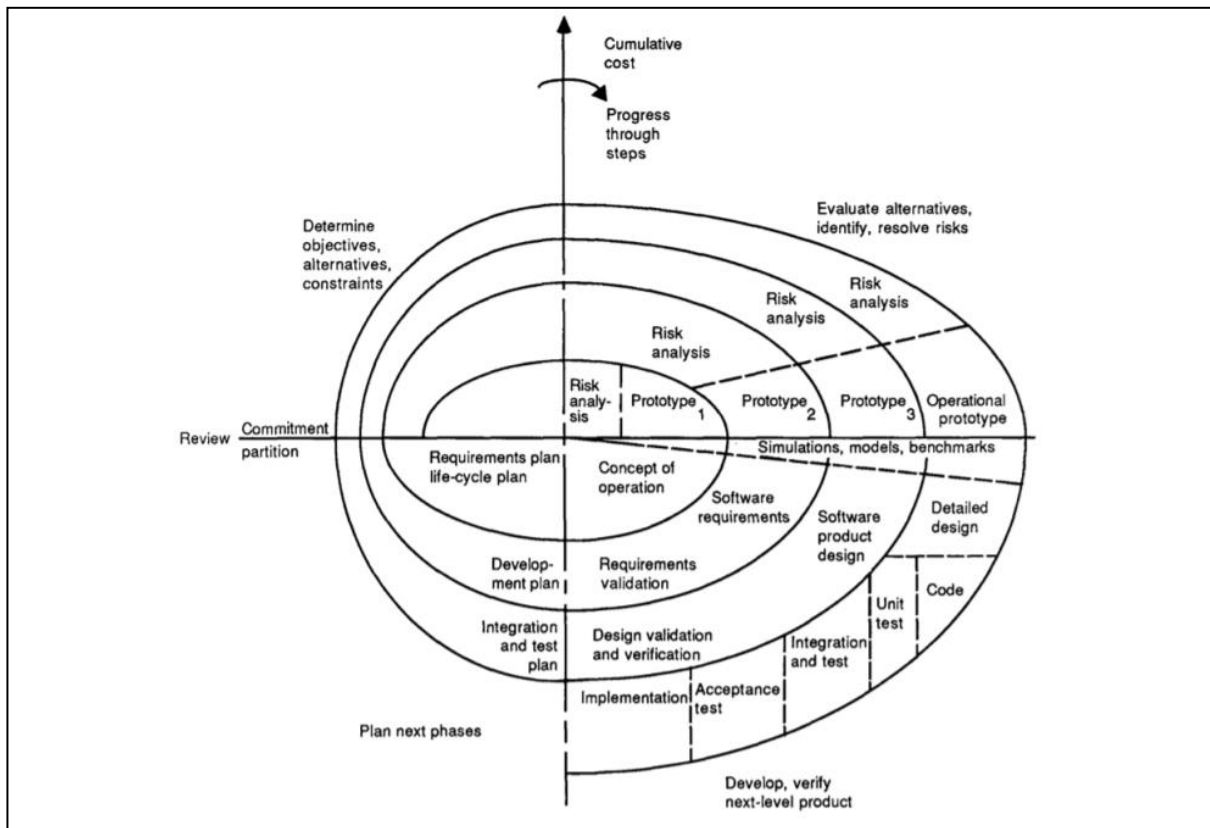


Figure 1. Spiral model as defined by Boehm [7].

These sections reflect the stages of the SDLC while also following the iterative design. Benefits of the spiral model include greater client collaboration due to multiple deployments and greater risk analysis due to the creation of multiple prototypes.

Agile Overview

The agile approach to software development is both iterative and incremental. Each completed development cycle is an iteration on the last and the product of each cycle is added together to incrementally build the system in full. The methodologies encapsulated by agile have defined steps similar to the SDLC but the priorities are different than traditional methodologies as described by the Agile Manifesto [8]. This manifesto is a document detailing the core principles behind agile which focus on the following:

- Interactions between real people rather than specific tools. Signifying that all parties have open communication to avoid misunderstandings rather than focusing on particular programming languages, infrastructure etc. [9]
- Less initial documentation and more focus on implemented software that works as intended. This principle represents the move away from traditional methods and more towards an iterative approach, where documentation, such as the SRS, is not needed and software is developed and deployed at a faster rate [9].
- Keeping the client involved throughout the development process rather than communication solely through meetings and contract negotiation. Creating a contract with the client is important for business and legal purposes, but it is much more beneficial for the client to be involved from the beginning as they can provide input and help guide the development to what they desire [9].

- Being adaptable to change rather than being bound to a rigid plan. Planning is incredibly important in software development but change is inevitable and so the development team must be able to adapt. Changes can take the form of technology advancements that could improve the production quality, or even changes in requirements from the client [9].

These main principles are not intended to refute the likes of defined processes, documentation and contract meetings, but for development teams to ask themselves what is more valuable to their projects [9]. The agile manifesto is a guideline for others to follow, not a strict set of rules, as such there are many subsequent methodologies that have been formed under the umbrella of agile [10], scrum is one such methodology that is discussed below.

Scrum – Description and Case Study

The most significant objective for scrum is to implement and deploy working software to the client as quickly as possible [11]. One of the most defining characteristics of scrum is sprints. Sprints are a length of time in which a full iteration of development takes place. A project will have a master list of requirements to complete known as the product backlog, which is populated by the leaders of the development team and the client. As the client is involved throughout the whole development process, requirements can be added to the product backlog at any stage. In a meeting at the beginning of each sprint, known as sprint planning, the development team discuss how many requirements they will be able to complete before the end of the sprint. These requirements move into a smaller list called the sprint backlog. The highest priority items in the product backlog are added to the sprint backlog as these are generally requirements that must be met before other features can be developed.

The requirements in a product or sprint backlog are called user stories. A user story is a short, commonly one sentence, statement describing an action that a user would want to perform when using the system [12]. A user can be a real person, such as an end user, but could also be another system that would build upon the implementation of a certain story. User stories have the following structure: As a <who>, I want <what>, so that <why> [13]. This structure provides clarity for the entire team of what each item in the backlog requires. User stories can be non-functional requirements, that is admin level requirements that the developers need in order to complete tasks, such as permissions or access to certain tools.

Scrum has several key personnel involved in the development cycle: the product owner, the scrum master and the development team [14]. The product owner focuses on understanding both the business and client requirements while prioritising the user stories in the product backlog.

The development team includes both software developers and testers and is responsible for creating the features for a sprint and ensuring they work correctly. A development team is comprised of a small number of developers (around five to seven) so to be able to distribute the workload but not struggle with communication. Each member of the team should be able to perform a number of tasks, ideally be able to perform any item in the backlog, to prevent an unfair distribution of work [14].

The role of the scrum master is to understand what their team is working on and to ensure everyone in the team understands the scrum ideas and principles [15]. They facilitate a smoother flow throughout each sprint by organising the various meetings of a sprint: planning, stand-ups, reviews and retrospectives [16].

Sprint planning is a meeting with all members of the team where they discuss what user stories they will be able to complete by the end of the sprint and how they will do so [17]. Once they have been agreed upon, the scrum master will move the stories into the sprint backlog and the team can begin working on them. Every day the scrum master holds a meeting with the development team called a

stand-up. In a stand-up each member physically stands up and briefly (around two minutes) describes what they did the previous day, what they will do that day and any obstructions they encountered [18]. If a member has an issue that needs discussing further, the stand-up is a good time to bring up the issue and then schedule a meeting with the appropriate person. Daily stand-ups are useful as everyone always remains on the same page and understands how the sprint is progressing. At the end of a sprint an informal meeting, known as a sprint review, is conducted. The sprint review is a time for the development team and any stakeholders to view a demo of what was developed during the sprint [14]. The product owner uses this meeting to review the items in the backlog and can make changes based on the feedback from the development team. A sprint retrospective is a meeting that allows the team to identify what went well and what did not go well as well as discuss ways the team can improve in the future [15].

Figure 2 shows the flow of processes in the scrum methodology.

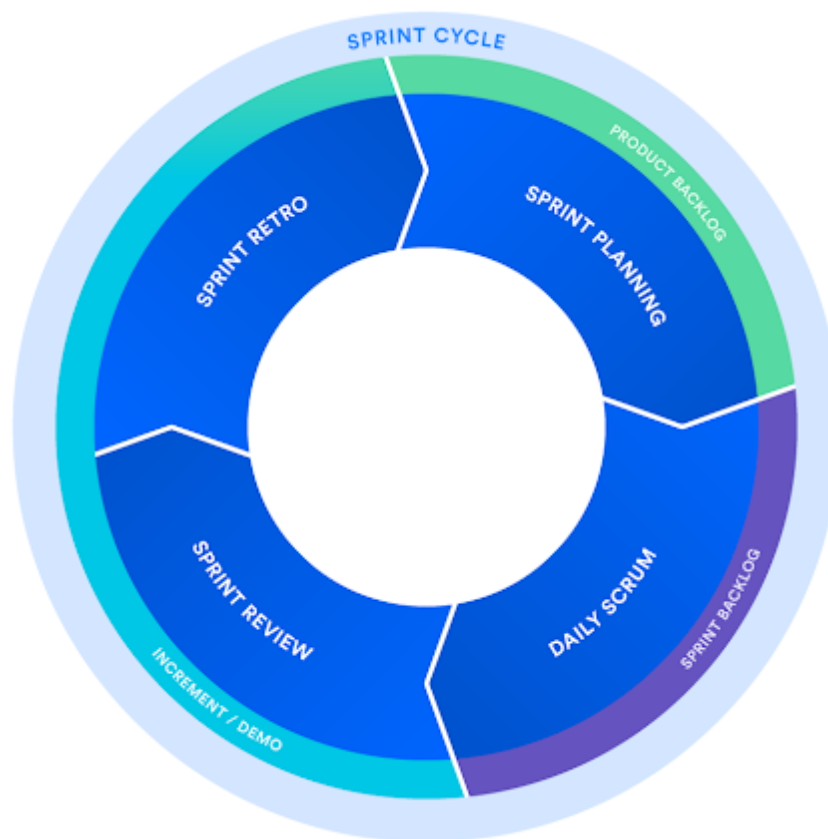


Figure 2. Process flow of scrum methodology [14].

When developing software, scrum is a great benefit as it provides constant communication between all parties and allows deployment of software periodically and more often than with traditional software methodologies. Scrum is a great benefit for the client, not just the business, as the client is involved throughout the entire development cycle, so they can see continuous process. A possible detriment that all scrum teams must be vigilant of is feature creep, as mentioned with the spiral methodology, this is usually a cause of the lack of initial documentation as well as the constant requirements refinements made by the product owner between sprints.

Case Study

One case study focused on a project for a Danish bank that utilised the scrum methodology [19]. The project consisted of two development teams one comprised of seven developers and the other of eight both with their own scrum masters and one product owner overseeing both teams. They conducted interviews and observations of the team in their working environment to evaluate how and why scrum works. The teams conducted daily meetings, similar to stand-ups, via conference calls as they were located in separate countries.

After conducting the study, the researchers found that scrum provided constant communication between team members fostering trust and coordination allowing for faster deployment and stronger code.

As a reverse to this, one case study showed that scrum can fail if communication and trust falter within a team [20]. This case study was performed over a period of nine months reviewing a team implementing scrum. The study found that one developer continued to develop features beyond the stories that were in the sprint backlog. When the scrum leader discovered this, a rift formed between the leader and the rest of the team. This led to the team not reporting problems when they occurred leading to less features in the sprint backlog being completed. As the leadership waned there was less communication within the team and planning for sprints become strenuous.

A flaw of scrum is that if communication is poor and leadership is weak then planning for sprints and completing user stories becomes very difficult and can cause a company money, or even cause a project to fail [20].

Conclusion

The agile approach to software development promotes communication and fast deployment of software through cyclical iterations. Though the agile approach follows the SDLC its methodologies focus more on working software than vast documentation. One methodology is scrum which a strong representation of the agile approach, fostering communication and team development through sprints, but can however, fail due to poor leadership and trust. These detriments are expressed in all agile approaches but are usually mitigated by the benefits. Ultimately agile is a better approach for smaller teams that are able to complete a project by deploying the software in stages.

References

- [1] N. Jain, "Software Development Life Cycle: A Detailed Study," *International Journal of Advanced Research in Computer Science*, vol. 2, no. 3, pp. 261-264, 2011.
- [2] A. Kazim, "A Study of Software Development Life Cycle Process Models," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 1, pp. 15-23, 2017.
- [3] Software Engineering Standards Committee of the IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998), Institute of Electrical and Electronics Engineers, Inc, 1998.
- [4] N. M. A. Munassar and A. Govardhan, "A Comparison Between Five Models Of Software Engineering," *International Journal of Computer Science Issues (IJCSI)*, vol. 7, no. 5, pp. 94-101, 2010.
- [5] P. Jalote, A. Palit, P. Kurien and V. Peethamber, "Timeboxing: a process model for iterative software development," *Journal of Systems and Software*, vol. 70, no. 1, pp. 117-127, 2004.
- [6] R. Larson and E. Larson, "Top five causes of scope creep ... and what to do about them," in *PMI Global Congress 2009*, Orlando, FL, 2009.
- [7] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, 1988.
- [8] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, "Manifesto for Agile Software Development," 2001.
- [9] M. Fowler and J. Highsmith, "The Agile Manifesto," Dr Dobb's, 01 August 2001. [Online]. Available: <http://www.drdoobs.com/open-source/the-agile-manifesto/184414755?queryText=the+agile+manifesto>. [Accessed 09 January 2019].
- [10] L. Williams, "What Agile Teams Think of Agile Principles," *Communications of the ACM*, vol. 55, no. 4, pp. 71-76, 2012.
- [11] N. Chantachaimongkol and P. Sincharoenpanich, "Critical factors for implementing the Scrum," 2013.
- [12] M. Trkman, J. Mendling and M. Krisper, "Using business process models to better understand the dependencies among user stories," *Information and Software Technology*, vol. 71, no. 1, pp. 58-76, 2016.
- [13] F. Dalpiaz and S. Brinkkemper, "Agile Requirements Engineering with User Stories," *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 506-507, 2018.
- [14] C. Drumond, "What is Scrum?," Atlassian, [Online]. Available: <https://www.atlassian.com/agile/scrum>. [Accessed 09 January 2019].

- [15] The Scrum Guide, "The Scrum Guide," The Scrum Guide, [Online]. Available: <https://www.scrumguides.org/scrum-guide.html>. [Accessed 09 January 2019].
- [16] M. Rehkopf, "What is a scrum master?," Atlassian, 11 July 2018. [Online]. Available: <https://www.atlassian.com/agile/scrum/scrum-master>. [Accessed 09 January 2019].
- [17] F. H. Cervone, "Improving Strategic Planning by Adapting Agile Methods to the Planning Process," *Journal of Library Administration*, vol. 54, no. 2, pp. 155-168, 2014.
- [18] V. Stray, D. I. Sjøberg and T. Dybå, "The daily stand-up meeting: A grounded theory study," *Journal of Systems & Software*, vol. 114, pp. 101-125, 2016.
- [19] L. Pries-Heje and J. Pries-Heje, "Why Scrum Works: A Case Study from an Agile Distributed Project in Denmark and India," in *2011 Agile Conference*, Salt Lake City, 2011.
- [20] N. B. Moe, T. Dingsøy and T. Dybå, "A teamwork model for understanding an agile team: A case study of a Scrum project," *Information and Software Technology*, vol. 52, no. 5, pp. 480-491, 2010.