

The Agile Software Development Approach

Abstract

The agile approach to software development is discussed, with reference to its origins and pre-existing frameworks. Agile methodologies are explored, with practical examples of use being given. Ultimately, this led to a positive conclusion regarding the potential benefits of adopting the agile approach.

Background

The Software Development Life Cycle

Prior to any attempt to discuss and contrast the different approaches to software development and the assorted methodologies associated with them, it is necessary to address the Software Development Life Cycle (SDLC) in general. This is because, rather than constituting a paradigm or methodology in itself, the SDLC can be more accurately described as an over-arching process or which has applicability to all of the different methodologies derived from both agile and traditional approaches [1].

According to Elliot [2], the SDLC came into existence during the 1960s as a result of attempts to develop information systems that would remain functional when applied to the sizeable business infrastructures of the time. However, this claim is contradicted by Kneuper [3], who suggests that the SDLC was first introduced by Herbert Benington in 1956, drawing on the work of hardware engineers working on comparable projects [4]. Although it is difficult to ascertain with great specificity the exact origin of the SDLC, it can be stated with some confidence that by the mid-1960s, the phases and sequence of the SDLC had been established.

The SDLC is represented in a multitude of different forms and is somewhat equivocal in its nature (as its purpose is essentially to be a representation of the stages of every development model). However, in essence, all illustrations of the SDLC contain the following sections: requirements analysis; design; coding, development or implementation; and testing or verification [5]. To summarise in less formal language; these steps represent finding out what the code needs to do, visualising and illustrating what the code should look like, writing the code and then making sure the code does what it's supposed to do. It is significant that some of the original depictions of what would later be known as the SDLC, such as Benington's contain all of the aforementioned phases, see figure 1 [6]. Although the order in which these stages are carried out might differ depending on approach and methodology, each of them will always be drawn on in some form, irrespective of whether the methodology falls under the umbrella of traditional, like the waterfall and v-shaped model or agile, like Scrum and Kanban [7].

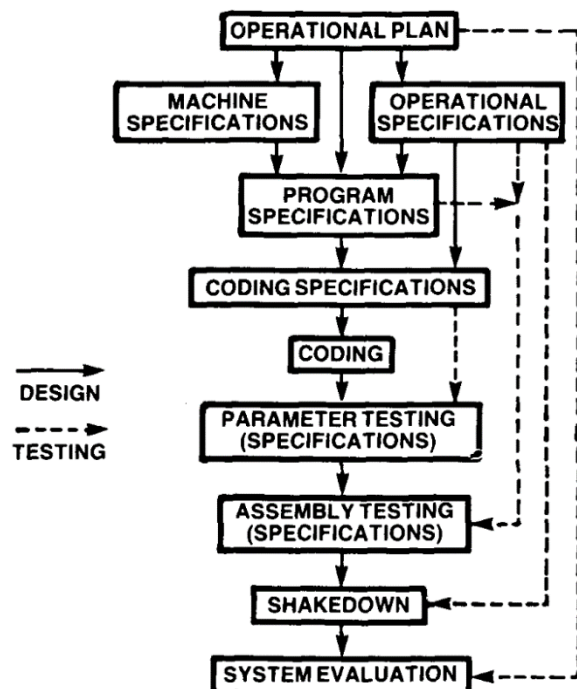


Figure 1: Benington's original SDLC from 1956 [6]

The Traditional Approach to Software Development

When the agile approach to software development is discussed, framed as a subset of the viable options, the natural question arises: “What, then, are the alternatives to agile, what isn’t agile?”. When addressing this issue, the predominant colloquialism is to subject all software to a binary choice: either it’s agile or it’s traditional. For reasons which it goes beyond the scope of this report to address, that depiction is, to a certain degree, inaccurate, with the consequent dichotomy being somewhat flawed. However, taking into consideration the relative advantage of brevity, for general purposes this classification is reliable, holding up to a sufficient level of scrutiny. This is because the most predominant of the non-agile methodologies put into practice today are those which can often be described as, in some senses, the predecessors of agile techniques, i.e. traditional methodologies [8].

Pre-eminent, or perhaps simply most familiar, amongst these processes is the waterfall methodology. A structure first attributed to the work of Royce in 1970, the waterfall methodology involves a linear, step by step process in which development progresses in a unidirectional fashion; starting with requirement analysis and culminating in testing and deployment [9]. It is relevant to note that although Royce is credited with the first illustration of the model in his paper and although he ultimately believed it to be “fundamentally sound”, he also pointed out some of the most significant flaws in the agile process, claiming it was “risky and invites failure” and drawing attention to the key issue of leaving testing to the end of the process [10].

Alternative methods, which still fall under the classification of traditional while attempting to better address issues such as the “compression of testing”, include the V-Model and the Spiral model. The V-model bears much similarity to the waterfall methodology, being purely incremental, however, it provides significantly more focus on the testing process, with each stage of development being associated with an equivalent testing procedure [11]. The spiral model, as laid out by Boehm [12], focuses on minimising risk, by planning out multiple successive cycles of software development, each spiral of which can be thought of as a complete run through of the waterfall process, see figure 2. The spiral model can be thought of as a move towards an approach which is both iterative and incremental and thus, in many ways, a move towards the type of software development process which would later be referred to as agile.

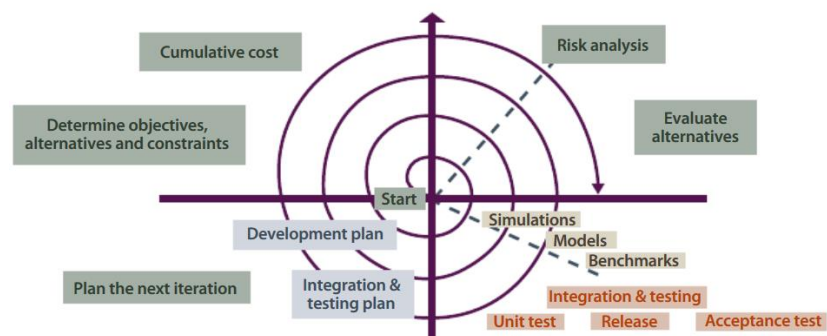


Figure 2: Boehm’s Spiral Model for software development [13]

What is agile software development?

Having illustrated the overall theme of the process which agile seeks to facilitate and having established with at least some degree of exactitude what agile isn’t, it then behoves one to ask the question: “What is agile?”

This question is more complex than first impressions would suggest. Rather than pinning the agile approach down to a particular type of process, or very specific group of methodologies, it might be more appropriate to describe it as a mind-set. This idea is supported by the Manifesto for Agile Software Development [13], the first argument of which values “individuals and interactions over processes and tools”, in other words, it’s more important to assemble a team which functions well rather than a team which functions in a very specific fashion. The agile manifesto goes on to place “working software over comprehensive documentation”, meaning that the extensive documentation produced by traditional methodologies such as waterfall, was less relevant as a criterion for success than the existence of the actual application which was meant to be developed [13]. Thirdly, the manifesto addresses the importance of “customer collaboration over contract negotiation”, identifying one of the common flaws in traditional approaches, where the customer is left out of the development process, with only a very strictly defined opportunity for input at the beginning [13]. Lastly, the authors of the agile manifesto called for flexibility and adaptability, valuing “responding to change over following a plan”, aiming to minimise

the possibility of getting to the deployment stage with a project that doesn't do what it is supposed to do [13, 14].

These four statements of relative worth have come to represent an entire approach, encapsulating numerous methodologies, which is defined under the term "agile". In short, if a methodology follows each of the stipulations of the agile manifesto, giving due weight to all the terms mentioned in it but attributing higher value to the first element of each statement, then it can, at least in some senses, be regarded as an agile methodology. However, it would be somewhat incorrect to suggest that there are not some more tangible common traits which can be recognised as maintained across the vast majority of agile methodologies. The most obvious of these is iteration, in order to follow the requirements of adaptability laid out in the manifesto, agile processes tend to work over a series of relatively brief time periods, which culminate in the release of an element of functionality which increases overall value, in this way, risk is minimised relative to the alternative of following a single, prescriptive, path from start to end [15].

So it is clear that one of the indicative characteristics for something being an agile methodology would be its implementation of an iterative process, with numerous short cycles of production. It is important to make a distinction here, it would be easy to categorise agile processes as being somehow free of the influence of pre-existing methodologies, metaphorically floating on the non-prescriptive cloud of the agile manifesto. However, to do so would be to remove a good portion of the context relevant to the agile approach. The manifesto itself was initiated in 2001 by 17 proponents of the lightweight software development processes such as Scrum, XP and FDD which were already in existence [16]. There is something of a mistaken chicken and egg scenario here, agile processes did not suddenly spring up after the agile manifesto, rather the agile-like processes in existence and evolution at that point in time, gave rise to the formal definition of "agile". As such, one should not deny the common roots which gave rise to certain threads of process similarity that are maintained to some extent across the multitude of agile processes implemented today.

Examples of Agile Processes

Scrum

Scrum is one of the most well-known and prolific of agile methodologies, having originated from a 1986 paper by Takeuchi and Nonaka [17], it was developed and formalised as a process over the course of the 1990s. As it exists today, Scrum is a fast paced, iterative and incremental process, which operates over a series of short "sprints" which are traditionally "timeboxed" to last no more than 30 days [18]. This time period is further divided, with each new day being heralded with a team catch-up session known as the daily stand-up, in which progress and goals are reaffirmed and reassessed. Each sprint aims to complete the "sprint-backlog", delivering an element of the required functionality, which is delineated by a set of "user stories", a set of customer-centred requirements, each one of which represents an attempt to add value to the final product. Over the course of multiple sprints, the Scrum team delivers on the "product-backlog", the compilation of all the user stories required to complete the product at full functionality, as shown in figure 3.

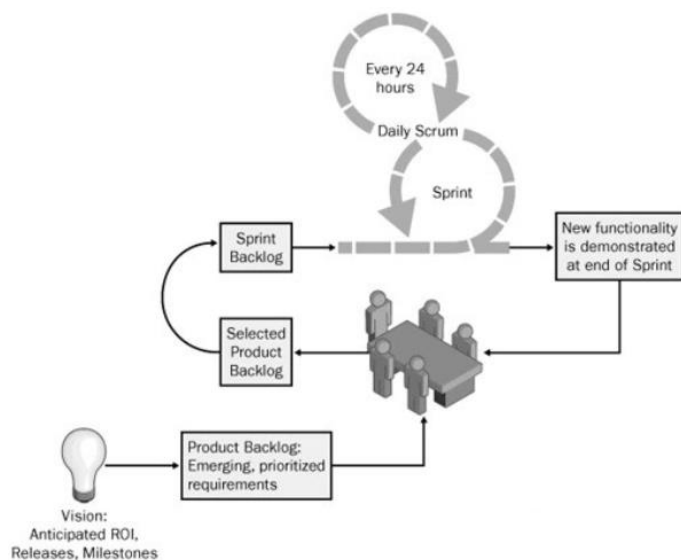


Figure 3: Scrum process overview [18]

In keeping with the ethos of agile, the product backlog and user stories are in a continual state of flux, with regular and in some senses continual user input being regarded as essential. As an agile process, there is practically no point in Scrum where the user requirements cannot be altered, to the extent that Schwaber [18], one of the key founders of Scrum, stated that "the product backlog is never complete".

Maintaining and prioritising this lengthy, ever changing list of user stories is the responsibility of the "Product Owner", one of the three positions typically allocated within the Scrum framework. The product owner is also

responsible for ensuring project consistency to a high level view of the end goals and ambitions, essentially acting as a customer proxy. Functioning as a counterpart to the product owner, the “ScrumMaster” is responsible for motivating and guiding the development team along the Scrum process. Acting as a “servant” to the team, it is the duty of the ScrumMaster to remove any “blockers” which might inhibit them in the completion of the sprint backlog [19]. The development team, of course, is responsible for adding value by coding and creating the functionality delineated by each user story, whilst also, due to the

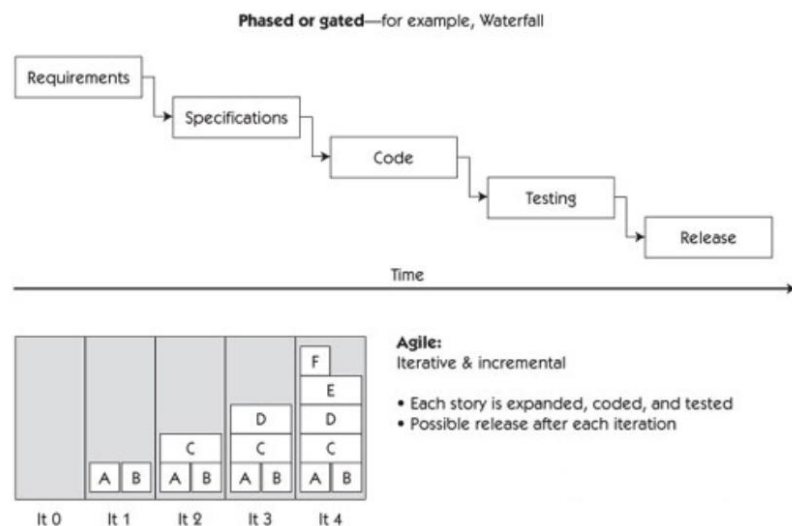


Figure 4: Lack of compression of stages in agile methodology such as Scrum vs traditional linear approach to software development [20]

iterative nature of Scrum, continually testing the code to ensure conformity with requirements. Often the development team will contain a set of members specialising in test, however each developer is responsible for providing some degree of validation and quality assurance on the functionality he has coded. As shown in figure 12, the development framework of Scrum prevents testing from being pushed into a very short period towards the end of the overall development process [20].

A practical example which demonstrates the efficacy of the agile Scrum methods, with particular relevance to the testing process, may be found in the development of the “telecommunication services delivery” management system of NetworkedAssets in 2013. An industrial case study of this mid-sized project by Berłowski et al. [21] found that adoption of agile Scrum practices led to favourable quantitative measurements of the readiness of the developing product to be released. This was particularly necessary in the specific context of this project, as although the Scrum squad was relatively small (consisting of 8 people), the development setting was somewhat intricate and unpredictable, with a high probability of impromptu release request. One other element specifically emphasised in this study was the benefit of the Scrum’s implicit focus, not merely on testing but also automated testing, which the team then expanded on, yielding a highly automated process which led to noticeable increased cost-effectivity of the undertaking [21].

Kanban

The Kanban methodology was synthesised from the JIT (Just-In-Time) manufacturing system that the Toyota automotive company utilised during the post-WW2 period [22]. Key to understanding the Kanban process is recognising its correspondence to the waste reducing objectives of the lean manufacturing process, which strives to eliminate inefficiency caused by: profligate use of resources, imbalanced work load and overburdening of team members [23]. Habitually, a Kanban board is used to maintain the flow of development work, keeping all team members up to date with progress and targets, whilst ensuring a continuous work pattern which nevertheless does not descend into overloading. This technique, derived from the original Toyota signboards (kanbans or 看板 in Japanese), contrasts with the timeboxed Scrum process, however the two may be combined into the composite methodology: Scrumban.

An example of a direct comparison between Kanban and Scrum can be found in a case study of Software Innovation, a company responsible for the production of document management software [24]. Employing approximately 100 developers, the business had benefited from the implementation of standard agile Scrum methods within their development teams. However, facing scaling issues, they elected to switch to the Kanban methodology, including the implementation of a Kanban Work in Progress (WIP) to avoid overloading and maintain a steady production process. Over the course of approximately two years the researchers noted an approximately 10% decrease in bugs and 20% increase in productivity, defined in this case by fulfilling requirement elements (User Stories) [24]. This would seem to suggest that in a given development environment, Kanban can prove more effective, not only than traditional approaches but also other agile methodologies such

as Scrum. However, the authors mention that familiarity with the agile approach, originating in the use of Scrum for two years prior to Kanban may have contributed to the observed improvements, so caution should be exercised before drawing sweeping conclusions from the above study alone.

Other Frameworks

Although Scrum and Kanban are prolific and well known amongst the array of agile methodologies, there are many more, such as the popular Extreme Programming (XP) and Rapid Application Development (RAD) frameworks. In essence, however, there are an almost uncountable number of possible software development processes which could conceivably be described as agile. Even apparently well-defined methodologies such as Scrum reveal a vast range of hidden variability, with every single software company implementing the fundamental concepts in slightly different ways. As such, although terms such as agile, traditional, Scrum and RAD are extremely useful in grouping together types of processes by the general characteristics by which they are applied, it would go far beyond the scope of this, and perhaps any, report to attempt to exactly classify every type of software development framework and determine the approach it utilises. One only need take the example of the UK's Driver and Vehicle Licensing Agency (DVLA), who most apparently utilise agile processes and specifically Scrum. In reality, however, the agency utilises a melange of Scrum, Kanban, XP and many other frameworks in a process which can only truly accurately be categorised as "The DVLA approach to software development".

Conclusion

The process of software development in general was discussed, with reference to the predecessors and alternatives to agile processes, before further investigation of the nature of the agile approach and an examination of the nature and consequences of some of the different methodologies attributed as being agile processes. Ultimately, there is significant indication that the adoption of agile frameworks can prove beneficial when compared with productivity under more traditional frameworks. This is indicated both by peer-reviewed case studies of the adoption of agile methodologies, albeit on the scale of individual companies, and by logical inference based on the analysis of the structure and format of the compared methodologies.

Works Cited

- [1] A. & B. A. Alshamrani, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model.," *International Journal of Computer Science Issues (IJCSI)*, vol. 12, no. 1, p. 106–111, 2015.
- [2] G. Elliott, "The traditional systems development life cycle," in *Global business information technology : an integrated systems approach*, Harlow, Pearson/Addison Wesley, 2004, pp. 86-125.
- [3] R. Kneuper, "Sixty Years of Software Development Life Cycle Models," *IEEE Annals of the History of Computing*, vol. 39, no. 3, pp. 41-55, 2017.
- [4] H. Benington, "Production of Large Computer Programs," *Annals of the History of Computing*, vol. 5, no. 4, p. 350–361, 1983.
- [5] N. Jain, "Software Development Life Cycle: A Detailed Study," *International Journal of Advanced Research in Computer Science*, 2(3, May 2011, Vol.2(3), vol. 2, no. 3, pp. 261-264, 2011.
- [6] H. Benington, "Production of Large Computer Programs (conf proc)," in *Symposium on advanced programming methods for digital computers*, Washington D.C., 1956.
- [7] 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.
- [8] H.-F. Chang and S. C-Y, "Toward the Integration of Traditional and Agile Approaches," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 2, pp. 9-13, 2013.
- [9] K. Petersen, C. Wohlin and D. Baca, "The Waterfall Model in Large-Scale Development," in *PROFES*, Olou, Finland, 2009.
- [10] W. Royce, "Managing the Development of Large Software Systems," in *Proceedings of IEEE WESCON*, 1970.
- [11] K. Forsberg and H. Mooz, "The Relationship of System Engineering to the Project Cycle," in *INCOSE International Symposium*, Chattanooga, 1991.
- [12] B. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14-24, 1986.
- [13] K. Beck, M. Beedle, A. V. 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," Agile Alliance, 2001 February 2001. [Online]. Available: <https://agilemanifesto.org/>. [Accessed 03 December 2018].
- [14] T. C. Krehbiel, P. A. Salzarulo, M. L. Cosmah, J. Forren, G. Gannod, D. Havelka, A. R. Hulshult and J. Merhout, "Agile Manifesto for Teaching and Learning," *Journal of Effective Teaching*, vol. 17, no. 2, pp. 90-112, 2017.
- [15] A. Moran, Agile Risk Management, Zurich: Springer Science & Business Media, 2014.

- [16] L. Williams, "What Agile Teams Think of Agile Principles," *Communications of the ACM*, vol. 55, no. 4, pp. 71-76, 2012.
- [17] H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Review*, vol. 64, no. 1, pp. 137-147, 1986.
- [18] K. Schwaber, *Agile Project Management with Scrum*, Redmond: Microsoft Press, 2004.
- [19] K. S. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process*, Boston: Addison-Wesley, 2012.
- [20] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*, Crawfordsville: Pearson Education, 2008.
- [21] J. Berłowski, P. Chruściel, M. Kasprzyk, I. Konanec and M. Jureczko, "Highly Automated Agile Testing Process: An Industrial Case Study," *E-Informatica Software Engineering Journal*, vol. 10, no. 1, pp. 69-87, 2016.
- [22] B. E, *Agile Project Management with Kanban*, Redmond: Microsoft Press, 2015.
- [23] J. Nicholas, "Race Without A Finish Line," in *Lean Production for Competitive Advantage: A Comprehensive Guide to Lean Methodologies and Management Practices*, Boca Raton, CRC Press, 2015, pp. 1-20.
- [24] D. SjøBerg, A. Johnsen and S. J. , "Quantifying the Effect of Using Kanban versus Scrum: A Case Study," *IEEE Software*, vol. 29, no. 5, pp. 47-53, 2012.
- [25] S. Groesser, A. Reyes-Lecuona and G. Granholm, "Extending the System Model," in *Dynamics of long-life assets: From technology adaptation to upgrading the business model*, Springer, 2017, pp. 169-189.