Historical Roots of Agile Methods: Where did "Agile Thinking" Come from?

Noura Abbas, Andrew M Gravell, Gary B Wills

School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
{na06r, amg, gbw}@ecs.soton.ac.uk

Abstract. The appearance of Agile methods has been the most noticeable change to software process thinking in the last fifteen years [16], but in fact many of the "Agile ideas" have been around since 70's or even before. Many studies and reviews have been conducted about Agile methods which ascribe their emergence as a reaction against traditional methods. In this paper, we argue that although Agile methods are new as a whole, they have strong roots in the history of software engineering. In addition to the iterative and incremental approaches that have been in use since 1957 [21], people who criticised the traditional methods suggested alternative approaches which were actually Agile ideas such as the response to change, customer involvement, and working software over documentation. The authors of this paper believe that education about the history of Agile thinking will help to develop better understanding as well as promoting the use of Agile methods. We therefore present and discuss the reasons behind the development and introduction of Agile methods, as a reaction to traditional methods, as a result of people's experience, and in particular focusing on reusing ideas from history.

Keywords: Agile Methods, Software Development, Foundations and Conceptual Studies of Agile Methods.

1 Introduction

Many reviews, studies and surveys have been conducted on Agile methods [1, 20, 15, 23, 38, 27, 22]. Most of these studies focus on the reaction to traditional methods as a reason behind Agile methods. However, Agile ideas have been around long time ago, and people who criticized the traditional methods suggested alternative approaches which were nothing but Agile ideas. Unfortunately these alternative approaches had not been treated seriously enough, and that is why it took us another 30 years to figure out that this is an effective way to develop software.

In their famous paper "Iterative and Incremental Development: A Brief History", Larman and Basili mentioned that iterative and incremental development was in use as early as 1957 [21]. In addition, they described projects that used iterative and incremental approaches in the 1970s. In this paper we will focus on the historical roots of other aspects of Agile thinking such as the response to change, customer involvement, and working software over documentation.

2 What Does it Mean to be Agile

The understanding of the word Agile varies in practice. In addition, it is difficult to define Agile methods as it is an umbrella for well-defined methods, which vary in practice. This section will show how this word was explained in literature by its proponents, as well as by other researchers.

Some researchers tend to define Agile as a philosophy. Alistair Cockburn's definition is "Agile implies being effective and manoeuvrable. An Agile process is both light and sufficient. The lightness is a means of staying manoeuvrable. The sufficiency is a matter of staying in the game" [13]. Barry Boehm describes Agile methods as "an outgrowth of rapid prototyping and rapid development experience as well as the resurgence of a philosophy that programming is a craft rather than an industrial process" [7].

Another way to describe Agile methods is by stating the basic practices various methods share. Craig Larman stated, "It is not possible to exactly define agile methods, as specific practices vary. However short timeboxed iterations with adaptive, evolutionary refinements of plans and goals is a basic practice various methods share" [22]. Boehm gives more practice-oriented definition, "In general, agile methods are very lightweight processes that employ short iteration cycles; actively involve users to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation" [7]. In an eWorkshop on Agile methods organized by the Centre of Experimental Software Engineering (CeBASE), the participants defined Agile methods as iterative, incremental, self-organizing, and emergent. In addition, they stated that all Agile methods follow the four values and twelve principles of the Agile Manifesto [15]. Boehm provided similar definition as he considered that a truly Agile method must include all of the previous attributes [7].

2.1 The Author's View

The previous reviews and discussions were essential to form our understanding of Agile methods. In this subsection, we will illustrate our understanding by providing our definition of an Agile method. In other words, what makes a development method Agile. An Agile method is adaptive, iterative and incremental, and people oriented

- **Adaptive**: an Agile method welcomes change, in technology and requirements, even to the point of changing the method itself [16]. In addition, it responds to feedback about previous work [22]. Fowler stated that an adaptive process is the one that can give control over unpredictability.
- **Iterative and incremental**: The software is developed in several iterations, each from planning to delivery. In each iteration part of the system is developed, tested, and improved while a new part is being developed. In each iteration, the functionality will be improved. In addition, the system is growing incrementally

as new functionality is added with each release. After each iteration (s), a release will be delivered to the customer in order to get feedback.

- **People-oriented**: "people are more important than any process. Good people with a good process will outperform good people with no process every time [8]. In an Agile method, people are the primary drivers of project success [13]. Therefore, the role of the process in an Agile method is to support the development team determine the best way to handle work [16]. Furthermore, an Agile method emphasises on face-to-face communication within the team and with the customer who is closely involve with the development process rather than written documents.

To summarize: Software development is an unpredictable activity; therefore, we need an adaptive process to control this unpredictability. Iterative and incremental development will be the best controller for this process. In addition, it needs creative and talented people.

3 What Was behind Agile Methods

Some interesting questions are:

What was behind Agile methods?

Where Agile methods were introduced?

What are the origins of Agile thinking?

We will answer these questions through three points: reaction to traditional methods and business change, reusing ideas from history, and people's experience. Then we will go through each Agile principle to see which one are new and which are not with evidence.

3.1 Reaction to Traditional Approaches and Business Change

Although iterative and incremental approaches were in use a long time ago, unfortunately, many sources still recommend the single pass software development lifecycle which known as the waterfall. However, researchers recognized the problem with the waterfall and suggested another approaches such as the V-Model [9], the Spiral model [6] and then the Rational Unified Process (RUP) [8]. These approaches tried to solve the waterfall problems but they are still heavyweight, document and plan driven approaches. Fowler refers to these approaches as engineering methodologies which may work perfectly for building a bridge but not for building software, as building software is a different kind of activity and it needs a different process [16]. Agile Methods are a reaction to the bureaucracy of the engineering methodologies. Another reason behind Agile methods is the increasing change in the business environment. According to Highsmith and Cockburn, Agile methods were proposed from a "perspective that mirror today's turbulent business and technology change" [13]. The traditional approaches could not cope with this change as they assume that it is possible to anticipate a complete set of the requirements early in the project lifecycle. In reality, most changes in requirements and technology occur within a project's life span.

3.2 Reusing Ideas from History

Many Agile ideas are hardly new. Furthermore, as the following paragraphs show many people believed long age that this is the most successful way of building software. However, these ideas have not been treated seriously, and in addition, presenting them as an approach for developing software is new [16, 22].

Iterative and incremental development is at the top of our list. When we defined an Agile methods we considered IID the heart of any Agile method. People were using these approaches successfully in the 70s and the 80s. Larman and Basili found early roots for (IID) iterative and incremental development since 1950s in NASA and IBM Federal Systems Divisions (FSD) [21]. According to them, NASA's 1961-63 Project Mercury was run with "short half-day iterations". In addition, the Extreme Programming practice of test-first development was applied as tests were planned and written and then the code were written to pass the tests. Furthermore, they use continuous integration as each mini-iteration required integration of all code and passing of the tests.

In 1970, Winston Royce who criticised the sequential model, recommended "five additional features that must be added to the basic approach to eliminate most of the development risks" [29]. These steps had the favour of iterative development. In step two, he recommended an early development pilot model for a 30-month project. This model might be scheduled for 10 months. In addition, in step five, he stated that the customer should be formally involved and he/she have to commit himself/herself at earlier points before the final delivery.

In their famous paper "Iterative and Incremental Development: A Brief History", Larman and Basili described a number of projects were iterative and incremental approaches were in use. These projects were major, government, life-critical systems, involving large numbers of people. In addition, most of the projects used a combination of top down concepts and incremental development. The projects used different iterations' lengths, which were longer than the range recommended by today's iterative methods.

People who criticised the waterfall suggested alternative approaches. In his paper "Stop the life-Cycle, I Want to get off", Gladden suggested a new view of the development process and he called it the Non-Cyclical Hollywood Model. According to Gladden, this model satisfies three propositions [18]:

System objectives are more important than system requirements: this meets the Agile idea of having a general understanding of the system rather than having detailed requirements which will change over the project

- A physical object conveys more information than a written specification: this is noted as the Agile manifesto values: Working software over Comprehensive documentation

- System objectives plus physical demonstrations will result in a successful product: by successful project he meant that a product that performs the function intended and satisfies the customer's need.

Gladden believed that most users do not have a clear idea about their needs. In addition, he raised the problem of missing and changing requirements.

Another suggestion was from McCracken and Jackson in their paper "Life Cycle Concept Considered Harmful". They suggested two scenarios of system development processes [25]:

Prototyping: They suggested building a prototype extremely early in the development process as a response to the early statements of the user. A series of prototypes or a series of modifications to the first prototype will gradually lead to the final product. This is exactly how development in Agile is meant to be, with short iterations each of which improves the system. In addition, they recommended a close relation with the user: "development proceeds step-by-step with the user, as insight into the user's own environment and needs is accumulated".

The second suggestion was a process of system development done by the end-user and analyst in this sequence: implement, design, specify, redesign, re-implement. Again, to start with implementing the system is the idea of modern iterative development. In addition, they suggested providing the user with an implementing tool and one version of a system. It is a similar idea of the CASE tools, which were used in Rapid Application Development (RAD) the early version on DSDM.

Agile ideas appeared in old development processes as well. In 1985, Tom Gilb wrote "Evolutionary Delivery versus the 'Waterfall model'". In this paper Gilb introduce the EVO method as an alternative of the waterfall which he considered as "unrealistic and dangerous to the primary objectives of any software project".

Gilb based EVO on three simple principles [17]:

- Deliver something to the real end-user
- Measure the added-value to the user in all critical dimensions
- Adjust both design and objectives based on observed realities

In addition, Gilb introduced his "personal list" of eight critical concepts that explain his method. When he discussed the early frequent iteration, he emphasised the concept of selecting the "potential steps with the highest user-value to development–cost ratio for earliest implementation" [17]. Another important concept in EVO method is "Complete analysis, design and test in each step" where he stated that the waterfall is one of the great time wasters with too many unknowns, too much dynamic change and systems complexity. Gilb stressed being user oriented:

"With evolutionary delivery the situation is changed. The developer is specifically charged with listening to the user reactions early and often. The user can play a direct role in the development process"

And being results oriented, not process oriented,

"Evolutionary delivery forces the developers to get outside of the building process for a moment, frequently and early – and find out whether their ship is navigating towards that port of call many cycles of delivery away"

Obviously, many of Gilb's concepts meet Agile principles. Not only the frequent delivery and the short iterations, but also he stressed the user role in the development process. In addition, he recommended an adaptive process and he gave the developers the power to change the direction of the process.

After Gilb's EVO, in 1988, the DuPont Company presented a methodology called Rapid Iterative Production Prototyping (RIPP). The main goal was to build working prototypes that could be presented to customers regularly to ensure that the finished product is what they wanted. The company guaranteed "Software in 90 days… or your money back" [3].

James Martin expanded this methodology into a large formalized one which became the Rapid Application Development (RAD). The RAD lifecycle has four phases: requirements planning, user design, construction and implementation (Martin 1991). What distinguishes RAD from traditional lifecycles is that in RAD construction phase we do the detailed design and code generation of one transaction after another. Each transaction can be shown to the end users to make adjustments. In addition, the "timebox" applies to the construction phase. The team will be given a fixed timebox within which the system must be constructed. The timebox inputs are the functions and the design framework of the system. The output is the system which will be evaluated to decide whether to put it in production or not. Within the timebox, "continuous iterative development is done" in order to produce a working system by the end of the timebox [24]. Martin recommended 60 days length for the timebox, with a 1-5 person team. The term "timebox" was created by Scott Shultz and was first used in DuPont. Shultz stated that the timebox methodology was successful as all the applications were complete in less time than it would have taken just to write the specification for a COBOL or FORTRAN application [24].

We can see that RAD has almost all Agile ideas. Actually, it formed the base for DSDM, one of the Agile methods [34]. RAD recommended quick delivery, iterative development, a small team of highly trained developers who work together at high speed, and user's involvement at every stage. Clearly, these ideas are the heart of Agile methods. However, the term "timebox" is used differently in Agile. In RAD, it is the whole construction phase and it consists of many iterations, where in Agile the timebox means a fixed iteration. In a fixed iteration, if the requests of the iteration can't be met within the timebox, the scope will be reduced [34,22].

3.3 People's Experience

As has been already mentioned, the manifesto gathered people who needed an alternative to traditional approaches. Importantly, most people involved in the manifesto had experience in software development. Furthermore, they had their own well-defined methods such as Extreme Programming (XP), Crystal and Scrum.

Ken Schwaber, one developers of Scrum, described his experience in the early 1990s when he was running a software company. He mentioned that their requirements were always changing and their customer's methodology did not help, instead it slowed them down. In order to solve the problem, he showed these methodologies to process theory experts at the DuPont Experimental station in 1995. He stated that they were amazed that his company was using an inappropriate process. In addition, they said that systems development had so much complexity and unpredictability that it had to be managed by an "empirical" process control model [32]. Ken's company and other organizations asked another question, which is why empirical development approaches deliver productivity while defined processes such as Capability Maturity Model (CMM) do not. They passed the question to scientists at

DuPont Chemical's Advanced Research facility, and the answer was that CMM is treated as a well-understood defined process while it is not, and it is performed without control and therefore it gives unpredictable results [31].

Kent Beck, founder of XP, also had a story. In April 1996, he was hired to help Chrysler, a payroll system. The project was in a state where two months away from production, the development team were not "computing the right answers yet". With the CIO of Chrysler, they decided to start from scratch with a smaller team. With Ron Jeffries, who became the first XP coach, and with the help of Martin Flower with analysis and testing, the first XP project took off. They worked on the base of three weeks iteration, where they implemented stories chosen by the domain expert. In April 1997, the system was live, and it was resalable, cheap and easy to maintain and extend. Beck stated "it was a technical and business success" [4].

Another story is from Alistair Cockburn, one of the Agile Manifesto authors. In 1991, IBM Consulting Group asked him to write a methodology for object-technology projects [12]. He decided to interview the project teams. He found out that their stories were different from what was mentioned in methodologies books. He found that "close communication, morale, and access to end users separated in stark contrast the successful projects [he] visited from the failing ones". Cockburn tried these ideas on a $15 million, fixed-price, and fixed-scope project of forty-five people. He was the lead consultant of the project and he wrote up the lessons learned from the project interviews, and from the project itself. Using these ideas, Cockburn built his Agile method Crystal. Interestingly, unlike most of other authors of the manifesto he stated that he came to Agile principles "through the need for efficiency, not the need to handle rapidly changing requirements".

3.4 What's New (and not) about Agile Methods

In this section we will go through each Agile principle, and we will try to find the roots of this principle. We will see this section will support our previous argument. This will be illustrated this in the next table.

Principle	New or not with Evidence
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	EVO first principle: deliver something to the real end-user [17]
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	Relatively new, the problem always existed but without a real solution
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	In EVO the frequent and early delivery is essential , also RAD recommended quick delivery [17, 24]
Business people and developers must work together daily throughout the project.	Relatively new as some approaches recommended good relation with customer, however, the idea of daily communication and on-site customer is new

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	These ideas were raised in the psychology of computer programming book which was published in 1985; the author empathized on the importance of motivation which is the inner directing force (chp10). In addition, he mentioned that the richness of the environment gives it a self–maintaining quality which resists the imposed changes.(chp4) [37]
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	The previous book focused on the importance of how the working space can affects the social interaction which in turn will affect the work. The author emphasized on how face to face communication helps transmitting useful information [37]
Working software is the primary measure of progress.	EVO second principle: measure the added-value to the user in all criteria dimensions [17]
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	In the Death March book, Edward Yourdon pointed the importance of managing and controlling progress and he suggested the “daily build” concept to succeed that mission [39]
Continuous attention to technical excellence and good design enhances agility.	Probably we could find the same idea of the importance of doing a much better programming job (technical excellence) in Dijkstra’s famous article “Humble programmer” [40]
Simplicity the art of maximizing the amount of work not done--is essential.	The famous saying on simplicity of design comes from Antione de Saint-Exupery: "Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away". [30]
The best architectures, requirements, and designs emerge from self-organizing teams.	We could find the idea of self-organizing team in open source projects which were out roughly at the same time as Agile methods. In the Cathedral and the Bazaar paper, Raymond referred to the developers as people bring their own resources to the table [28]
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.	The idea of process improvements was presented in CMMI level 5 with different emphasize as in Agile all the team will reflect on improving the process not only the management [36]

4 Discussion and Conclusion

Although Agile methods are new as a whole, their principles and ideas have been around long time ago, and people who criticized the traditional methods suggested

alternative approaches which were nothing but Agile ideas. Unfortunately these alternative approaches had not been treated seriously enough. For example Somerville first edition of the software engineering book describes “The Software Lifecycle”. At this point the word “waterfall” was not yet in common use: if you assume there is only one lifecycle, you do not need to give it a name. By the time edition 1989 Sommerville states that “one of the reasons for the wide spread adoption of the ‘waterfall’ model is that it allows for the straight-forward definition of milestones throughout the course of a project. Alternative approaches, such as evolutionary prototyping, are such that milestone definition is a more difficult and less certain process”. Even in the most recent edition (the 8th in 2007) Sommerville devotes just one chapter (chapter 17) out of 32 to “rapid” software development. In this chapter it is claimed that “dissatisfaction with these heavyweight approaches led a number of software developers in the 1990s to propose new agile methods” [43]. In this paper we provided historical and anecdotal evidence that a) dissatisfaction with heavyweight approaches existed long before the 1990s, b) non-waterfall projects succeeded as early as 1957 and c) viable alternatives such as EVO, RAD and RIPP had been developed and applied successfully in the 1980s.
We hope that the strong emerge of Agile methods and the pressing need for such development methods these days will convince the software development community that this is the right way to develop software. In addition, we think that the education about Agile thinking history will help understanding as well as promoting the use of Agile methods.

References

1. Abrahamsson P., Solo O., Ronkainen J. and Warsta J. (2002). Agile Software Debvelopment Methods, VTT technical Research Centre of Finland.
2. Ambler S. (2005). "Quality in an Agile World." Software Quality Professional 7(4): 34-40.
3. Ambrosio J. (1988). Software in 90 days Software Magazine, Wiesner Publications, Inc.
4. Beck K. and Andres C. (2004). Extreme Programming Explained: Embrace Change (2nd Edition), Addison-Wesley Professional.
5. Boehm B. (1979). Guidelines for Verifying and Validating Software Requirements and Design Specifications, Euro IFIP 79, P. A. Samet (editor), North-Holland Publishing Company, IFIP.
6. Boehm B. (1988). "A spiral model of software development and enhancement." IEEE Computer 21(5): 61-72.
7. Boehm B. and Turner R. (2003). Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley Longman Publishing Co., Inc.
8. Booch G. (1995). Object Solutions: Managing the Object-Oriented Project, Addison Wesley Longman Publishing Co., Inc.
9. Coad P., deLuca J. and Lefebvre E. (1999). Java Modeling Color with UML: Enterprise Components and Process with Cdrom, Prentice Hall PTR.
10. Cockburn A. (1999). Characterizing People as Non-linear,First-Order Components in Software Development, Humans and Technology Technical Report.
11. Cockburn A. (2002a). Agile Software Development, Addison-Wesley Longman Publishing Co., Inc.

12. Cockburn A. (2005). Crystal Clear A Human -Powered Methodology for Small Teams, Addison-Wesley.
13. Cockburn A. and Highsmith J. (2001a). "Agile Software Development: The Business of Innovation." Computer 34(9): 120-127.
14. Cockburn A. and Highsmith J. (2001b). "Agile Software Development: The People Factor " Computer 34(11): 131-133.
15. Cohen D., Lindvall M. and Costa P. (2004). "An Introduction to Agile Methods." Advances in Computers: 1-66.
16. Fowler M. (2005). The New Methodology, www.martinfowler.com.
17. Gilb T. (1985). "Evolutionary Delivery versus the "waterfall model" " ACM SIGSOFT Software Engineering Notes 10(3): 49-61.
18. Gladden G. R. (1982). "Stop the Life-cycle, I Want to Get off." 7(2): 35-39.
19. Highsmith J. (2000). Adaptive Software Development: a Collaborative Approach to Managing Complex Systems, Dorset House Publishing Co., Inc.
20. Highsmith J. (2002). Agile Software Development Ecosystems, Addison-Wesley Longman Publishing Co., Inc.
21. Larman C. and Basili V. R. (2003). "Iterative and Incremental Development: A Brief History." IEEE Computer Society 36(6): 47-56.
22. Larman C. (2004). Agile and Iterative Development: A Manager's Guide. C. Alistair and H. Jim, Pearson Education, Inc.
23. Levine L. (2005). Reflections on Software Agility and Agile Methods, Software Engineering Institute Carnegie Mellon University Pittsburgh, PA U.S.A.
24. Martin J. (1991). Rapid Application Development, Macmillan Publishing Co., Inc.
25. McCracken D. D. and Jackson M. A. (1982). Lifecycle Concept Considered Harmful, ACM Press. 7: 29-32.
26. Palmer S. R. and Felsing M. (2001). A Practical Guide to Feature-Driven Development, Pearson Education.
27. Paulk M. C. (2002). "Agile Methodologies and Process Discipline." CrossTalk- The Journal of defence Software Engineering: 15-18.
28. Raymond, E.S., The Cathedral and the Bazaar. 1999.
29. Royce, W.W., Managing the Development of Large Software Systems 1970, Proceedings, IEEE WESCON p. 1-9.
30. Saint-Exupery, A.d., Wind, Sand and Stars. 1992: Harcourt
31. Schwaber K. (1996). Controlled Chaos : Living on the Edge, Advanced Development Methods, Inc.
32. Schwaber K. and Beedle M. (2001). Agile Software Development with Scrum, Prentice Hall PTR.
33. Sommerville I. (1972)(1989)(2007). Software Engineering (1st, 3rd, 8th) Edition, Addison-Wesley.
34. Stapleton J. (1997). DSDM: The Method in Practice, Addison-Wesley Longman Publishing Co., Inc.
35. Thomas D. (2006). The Essential Unified Process (EssUP) - New Life for the Unified Process Dr. Dobb's Portal: the Worlds of Software Development.
36. Turner, R. and A. Jain, Agile Meets CMMI: Culture Clash or Common Cause? . 2002, XP/Agile Universe 2002, LNCS 2418 p. 153-165.
37. Weinberg, G.M., The Psychology of Computer Programming. 1985: John Wiley \&amp; Sons, Inc. 304.
38. Williams L. (2004). A Survey of Agile Development Methodologies.
39. Yourdon, E., Death March: The Complete Software Developer's Guide to Surviving Mission Impossible Projects, ed. D.B. Paul. 1997: Prentice Hall PTR. 227.
40. Dijkstra, E.W., The humble programmer. Communication of the ACM, 1972. 15(10): p. 859-866.