The Advantages of Agile Development

John Foley

13 April 2014

Abstract

There are many problems in software engineering that Agile Methodologies are designed to address, such as ability to react to change and adapt, maintain communication with clients instead of renegotiating contract, and evolutionary development cycles instead of planned in phases. These solve problems that are inherent in software development, and are not handled well in past methodologies such as Waterfall.

1 Introduction

Agile Methodologies have taken the software engineering industry by storm, especially in this modern era of internet technology and application production. The Agile Manifesto, published in 2001, introduced many terms and ideas to describe the current set of methodologies that we call Agile. Unlike it's predecessors, Agile focuses on non-deterministic, complex systems where its difficult to start designing in the beginning. After years of successes and failures, software developers began seeing that upfront designs of these type of systems only lead to divergence, massive overhauls, and as a consequence produce overwhelming waste. The type of strategy that waterfall for example follows is primarily predictive. Client requirements for the system are determined and processed, and then different phases of design and implementation are strictly followed. Each phase is built on the previous and attempt to predict the future sections of the system that will be addressed in the next phase, until the application is ready to be implemented. This layered style involves traceable documentation and designing every aspect of the application in one long process. Agiles core philosophy is to take the opposing face of the same coin and be adaptive instead of predictive. The same long, phased process that waterfall methodologies advocate are instead broken up into smaller, easier to handle chunks.

Since software development typically takes one to several years, a development methodology

must be able to organize and deal with time and money constraints. Agile deals with this problem by introducing iterations, and an iteration is a method of breaking time and features up into workable, adaptive slices. Each series of features or changes are called sprints, each iteration is meant to move the systems development up one step closer to being completed. Each iteration is suppose to be independent enough to not worry about the next iteration, and thus not predict anything because change is actually encouraged in agile methodologies. Change is so expected that agile actually works at its best when the ideas of the application change after each iteration. This aspect makes agile incredibly effective in non-deterministic, complex systems where features are complex and need to be modified with each new addition.

Large systems development requires collaboration between teams and departments in order to complete the product. This leads to another core aspect of Agile, which is the small team approach. By keeping teams tight and cross functional, communication is abundant and the project is easier to manage. Communication isnt kept exclusive to teammates; clients or client domain representatives are welcomed to be in the same room so that developers can ask questions and maintain convergence with the system that the clients desire. This communication keeps development adaptive and the teams on track with an accurate end product.

Requirements and constraints are likely to change over time, so a methodology must be able to cope with change over the years of its life cycle and maintenance beyond that. Iterations and constant adaptation leads to the final core philosophy of agile methodologies, evolutionary design. Every iteration organizes and pushes the project up, and can be thought of as a generation. A generation is independent and is expected to be operational by itself, even if it doesnt satisfy every feature and requirements. In fact, it isnt suppose to. Every generation is built on the previous and expected to be better or expanded in some way, and changes can be made easily and quickly.

Procedural methodologies that have been used for years have attempted to cope with these problems in a variety of ways. Waterfall tries to predict and document for every aspect of the application that it can, and does a great job if the project is sequential and any after-the-fact changes are considered prohibitively costly. In a modern era of software development, new methodologies are required to deal with the growing significance of department collaboration, evolutionary design, and long development durations. Agile has been produced to manage these problems.

2 Client Contact with Small, Cross-Functional Teams

Communication with the client during the life of software is critically important. It ensures that the product matches what the client is expecting. In the past, client requirements are constructed into a contract for the developer to create, but that allows for divergence if every detail is ambiguous. Agile Methodologies that teams and a client representative create a relationship to stimulate communication.

2.1 Client in the Room

Agile Methodlogies advocate for someone with domain knowledge be present during development so that specific details and questions can be answered immediately. Of course this would be hard and costly to maintain, so good communication should be maintained in its stead.

2.2 Small Teams and Scrums

Teams are kept small in order to maintain inter-team communication. Internal workings are easier to maintain and manage, as well as becoming more comfortable with whom a developer works with.

2.3 Scrums and Meetings

Scrums are quick, efficient meetings for teams to keep on track. Scrums are held often and typically report driven.

3 Iterative Cycles

One strength of the Agile Methodology is how the software development life cycle is separated into easier to handle segments. Agile thinks of the time it takes to solve a specific feature or problem as an iteration, typically two weeks long, but is up to the production manager. This allows for

structured development of the application, and breaks up the requirements from the user into smaller, easier to handle chunks.

3.1 Iterations

The structured phases of feature and problem development of the application

3.2 Separation into Easier Phases

Each iteration allows for the application to be developed in parts. This allows for flexible changes and extremely reactive development. [**?**]

4 Evolutionary Development

4.1 Test Driven Development

The use of Test Driven Development allows for an application that ensures that past work done on the application is still working with the addition of new code.

4.2 Increment

Each iteration has a specific goal in mind. After each iteration is complete, the application is one step further, and considered incremented forward.

5 Past Methodologies

The Waterfall Methodology has been the primary methodology used for software development for decades, and is still used today. These methodologies are effective, but do not address major problems that lead to either failure of a software development project or shortcomings of an application.

5.1 Phases Slow Reaction Time

Waterfall is typically held with several phases, one of which is the design phase, followed by implementation phase. These phases are essentially "closed door" meetings for developers from clients, and so divergence and thus failure to respond to change is a problem.

5.2 Maintain Communication

Waterfall is not known to maintain strong client communication, and without constant updates and checks on progress, the project will inevitably diverge from what the client is asking for.

5.3 Evolution of the Product

Waterfall methodologies advises development to be carried out in phases, without clear benchmarks to be made. A common problem is that the project is developed all at once and changing subcomponents becomes difficult and overwhelms a project.

6 Conclusion

References

[1] Lan Cao, Balasubramaniam Ramesh, and Tarek Abdel-Hamid. Modeling dynamics in agile software development. *ACM Trans. Manage. Inf. Syst.*, 1(1):5:1–5:26, December 2010.

[2] André Janus. Towards a common agile software development model (asdm). *SIGSOFT Softw. Eng. Notes*, 37(4):1–8, July 2012.

[3] Oualid Ktata and Ghislain Lévesque. Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects. In *Proceedings of the 2Nd Canadian Conference on Computer Science and Software Engineering*, C3S2E '09, pages 59–66, New York, NY, USA, 2009. ACM.

[4] Magnus Thorstein Sletholt, Jo Hannay, Dietmar Pfahl, Hans Christian Benestad, and Hans Petter Langtangen. A literature review of agile practices and their effects in scientific software development. In *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering*, SECSE '11, pages 1–9, New York, NY, USA, 2011. ACM.

[5] Christoph Johann Stettina and Werner Heijstek. Necessary and neglected?: An empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM International Conference on Design of Communication*, SIGDOC '11, pages 159–166, New York, NY, USA, 2011. ACM.