International Journal of Engineering, Science and Metallurgy

(General article)

An Empirical Study of Agile Software Development

Ajay Deep[a*]

Department of Information Technology, OITM, Hisar

Accepted 11 July 2012, Available online 1Sept 2012

Abstract

For Developing software there are two main methodologies: the traditional sequential or "the waterfall method" and the iterative or "The Agile Method". In this paper we emphasis on Agile Manifesto and its principles. It provides the general understanding of Agile Software Development and can act as a foundation for choosing methodology for the software projects as we have compared different methods and listed their Applicability and Limitations.

Keywords: *Agile, Agile manifesto, Principle of Agile manifesto, Usability, Waterfall Method.*

1. Introduction

The agile methods analyzed and reported in this paper are those consistent with the Manifesto for Agile Software Development. Now a day, software project management is becoming more and more important since a project needs an organized plan to follow through. There are two famous process models in this area, which are the traditional Waterfall Process Model and the Agile Software Development. The Agile software development techniques are gradually being accepted as viable alternative to traditional software development methodologies. It leads to better quality software in a shorter period of time.

a) Waterfall model

In Royce's original waterfall model, the following phases are followed in order: Analysis, Requirements Specification, Design, Implementation, Testing and Integration, Operation and Maintenance. The *waterfall model* proceeds from one phase to the next in a sequential manner. For example, one first completes requirements specifications, which after sign-off are considered "set in stone". When requirements are completed, one proceeds to design. The software in question is designed and a blueprint is drawn for implementers (coders) to follow—this design should be a plan for implementing the requirements given. When the design is complete, an implementation of that design is made by coders. Towards the later stages of this implementation phase, separate software components produced are combined to introduce new functionality and reduced risk through the removal of errors

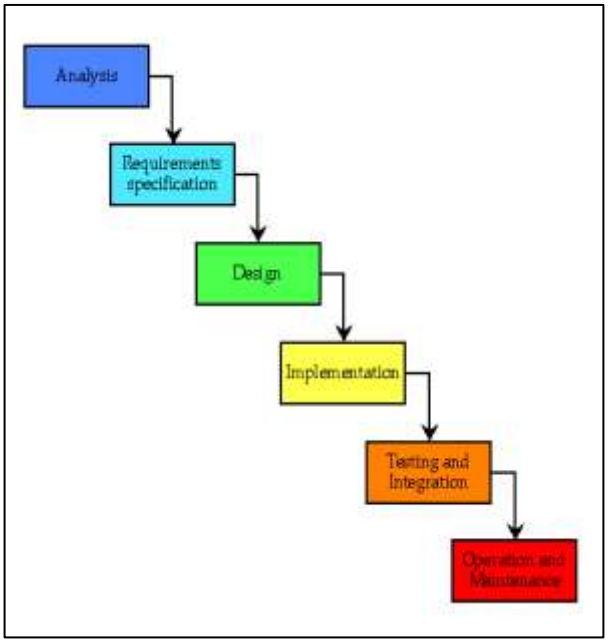

Figure1: Waterfall Model

b) Agile Model

Adaptable software creation, also known as agile software development. Agile development is a style of software development that emphasizes customer satisfaction through continuous delivery of functional software. Based on a variety of iterative development disciplines including

* Corresponding author's email: er.ajaydeep@gmail.com

extreme programming (XP), agile methods put developers to work in small teams to tight budgets and short timescales. In contrast to traditional software development methods, agile developers liaise continuously with business clients, aiming to deliver working software as frequently as every two weeks during a project, and welcome changes to the requirement in response to evolving business needs. The concept of the Waterfall Process Model is that the requirement analysis has to be done in the beginning phase, whereas, the Agile Software Development emphasizes that the requirement is changeable throughout the process. Thus, it seems that using the Agile Software Development is becoming a trend for software development companies in order to improve the software process.

Agile is a group of software engineering methodologies, e.g. eXtreme programming , Scrum , Crystal that aim to increase overall software developer productivity, deliver working software on time, and minimize the risk of failure in software projects. Software is developed through short time boxed cycles, typically of one to four weeks in duration, known as iterations. Each iteration is, a mini-software project encompassing planning, requirements analysis, design, coding, and testing. The output is working software that integrates successfully with the working software developed in the preceding iteration. Software is released after a series of iterations, typically every few months.

1. Agile Manifesto

The Agile Manifesto stresses the importance of:-
a) People and interactions over processes and tools,
b) Working software instead of detailed documentation,
c) Active customer participation and involvement rather than time and effort expended on negotiating contracts, and
d) Willingness and ability to take on changes over steadfast commitment to a static plan.
Agile software development methods including eXtreme Programming (XP), Scrum, Adaptive Software Development and Feature-Driven Development are based on the principles of the Agile Manifesto and geared towards realizing its goals and objectives. In general, the feedback from organizations that have implemented agile development is positive.

a) Principles of Agile Manifesto

The values described above are realized in the principles of Agile Manifesto. The principles are the following:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

b) Background and values of Agile Manifesto

Agile Alliance was formed when seventeen representatives of different agile methods, such as Extreme Programming (XP), Scrum and Crystal Family, met to discuss alternatives to rigorous, documentation driven software development. Agile Alliance did not want to specify detailed project tactics mainly because the members represent competing companies but rather agree on values that support agile or lightweight software development at a high level. Thus, Agile Manifesto is a collection of values and principles, which can be found in the background of the most agile methods. Agile Alliance describes its intentions as follows:
"Agile movement is not anti-method, in fact, many of us want to restore credibility to the word method. We want to restore a balance. We embrace modeling but not in order to file some diagram in a dusty corporate repository. We embrace documentation but not hundreds of pages of never- maintained and rarely used tomes. "
Agile Alliance formulated their ideas into values and further to twelve principles that support those values. Values of Agile Manifesto are the following:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

c) Key Ideas for Agile Software Development Methodologies

The keys for successful use of choosing and using agile methodologies are summarized:-

- Agile methods are a subset of iterative and evolutionary methods. Iterations are short to provide for more timely feedback to the project team.
- The Agile Manifesto documents the priorities that underlie the principles and practices of agile software development methodologies.
- Extreme Programming is based upon four values and 12 specific software development practices.
- The Crystal family of methodologies is customizable based upon the characteristics of the project and the team.
- Scrum mainly deals with project management principles. The methodology allows the team freedom to choose its specific development practices.
- FDD methodology has the most thorough analysis and design practices.
-

1. The Agile Project Lifecycle

As mentioned, the Agile Development Framework is an iterative, incremental, and collaborative methodology for software development project. The Figure shows that the Agile Software Development Lifecycle (ASDL) starts from an initial plan and ends with deployment. Each iteration consists of planning, requirements, analysis and design, implementation, deployment, testing, and evaluation.

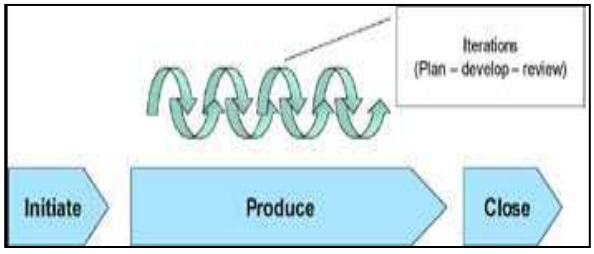

Figure 2 Agile Project Life Cycles.

The Agile Software Development was developed to solve the Waterfall's weakness. Differing from the Waterfall Process Model, the Agile Software Development emphasizes its flexibility. For example, requirements are allowed to change in an agile project. A face to face communication also takes an important place in the Agile Software Development since it requires a quick feedback.

a) General Understanding of the Agile Software Development

Generally, agility is defined by ability which is able to be flexible and adaptable to change. The idea of the Agile Development Framework is to create a pain-free working environment for those small, co-located, self-organized teams in order to assist companies to take full advantage of the customer value of the delivered software product.

On an Agile project, developers work closely with their customers to understand their needs, they are placed in a pair to implement and test their solution, and the solution is shown to the customers for quick feedback. Therefore, the business contract will not become a barrier between customers and developers, but a platform to help customers and developers work together.

Moreover, the Agile Software Development is the work of energizing, empowering, and enabling project teams to rapidly and reliably deliver business value by engaging customers and continuously learning and adapting to their changing needs and environments.

In the Agile Development Framework, user requirements are written from users' perspective, elaborating on what users want to do with a feature of the software. The concept of the Agile Software Development can be summarized as below:

Eliminate Waste: The Agile Development Framework advocates a "barely sufficient" approach to plan, process, and control software development process. "Barely sufficient", in other words, is to find the simplest things to meet needs of requestor instead of wasting unnecessary resources.

Sustainable Pace: The Agile Development Framework requires a daily meeting. All team members have to report what they have accomplished and what they are going to do in the meeting so that the progress of the project can be traced.

Intense Collaboration: Unlike the traditional approach, which relies on documents, the Agile Development Framework requires a daily face to face communication with customers and coworkers to understand and fulfill their requirements.

Frequent Delivery: Frequent delivery offers incremental business value to customers. Customers experience the growth of the system and obtain additional insight to how requirements are planned by interacting with the system early. Thus, frequent delivery is one important way to seek feedback on the quality of the application.

Continuous Feedback: The Agile Development Framework was invented to achieve customer's value. Thus, continuous feedback is demanded in order to inspect if the development team is well aligned with business objectives.

Include Change: Differing from the traditional approach, change of requirements is considered in the Agile Development Framework since remaining adaptable is a key to building a trusting relationship with customers. Furthermore, prioritizing features, exploring and explaining risk help both of team members and customers understand the consequence of change.

Empowerment: The Agile Development Framework also pays attention to the team working atmosphere. It is very important to set-up a well-organized and positive team. Therefore, encouraging team members is highly required.

Iterative and Incremental development: plans, requirements, design, implementation, deployment and testing are developed incrementally through iterations. Each iteration usually takes two to four weeks. Moreover, problem hunting, scope solving, feedback collection should be done at the end of each iteration. In addition, features and tasks are also inspected and tracked within each iteration.

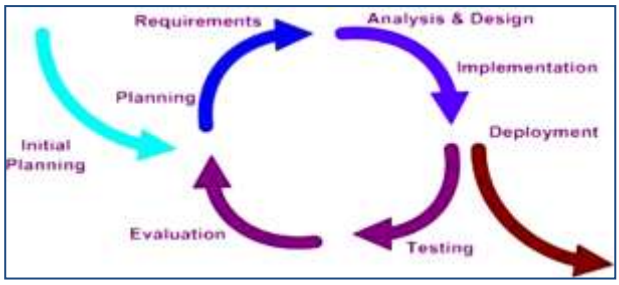

Figure 3: Iteration in the Agile Development Framework.

Establish and Changing Requirements: In traditional software development process model, identification and analysis of requirements for the system are documented within an agreement for customers and the development team in an early phase of a project. Once this agreement has been reached, the requirements are not allowed to change. In contrast, the Agile Development Framework allows both customers and developers to change the requirements throughout the project, but only the customers have the authority to approve, disapprove and prioritize the ever changing requirements. In addition, the requirements can be reprioritized anytime. Once the priority changed, the new higher requirement will be pulled up to the top of the stack.

Use Backlog: In the Agile Development Framework, tasks are divided into small chunks (also known as backlogs) to manage complexity and to get quick feedback.

Prioritize Backlog: In an Agile team, backlogs are prioritized by customers and only higher prioritized backlogs (top 2 or 3) will be processed in iteration. Once the backlog prioritizing works are done, the prerequisites for calling the first Sprint Planning Meeting will be embraced. Moreover, unfinished backlogs are inspected and reprioritized at the end of each iteration in order to decide which backlogs will be processed in the next iteration.

Face to Face Communication: The Agile Development Framework emphasizes face to face communication. It promotes holding a short daily meeting. In the meeting, team members have to report their project progress.

Pairing Programmer & Self-Organization: In an Agile team, a less experienced programmer is paired with an experienced one in order to share knowledge and teach each other. Moreover, each Agile team is self organized. Team members are self organized by accomplishing tasks with their co-works from backlogs.

Limitations of Agile Processes

The assumptions listed above do not hold for all software development environments in general, nor for all “agile” processes in particular. In this part I describe some of the situations in which agile processes may generally not be applicable. It is possible that some agile processes fit these assumptions better, while others may be able to be extended to address the limitations discussed here. Such extensions can involve incorporating principles and practices often associated with more predictive development practices into agile processes.

a) Limited support for distributed development environments:

The emphasis on co-location in practices advocated by agile processes does not fit well with the drive by some industries to realize globally distributed software development environments. Development environments in which team members and customers are physically distributed may not be able to accommodate the face-to face communication advocated by agile processes.

b) Limited support for subcontracting:

Outsourcing of software development tasks to subcontractors is often based on contracts that precisely stipulate what is required of the subcontractor. Subcontracted tasks have to be well-defined in the cases where subcontractors have to bid for the contract. In developing a bid a subcontractor will usually develop a plan that includes a process, with milestones and deliverables, in sufficient detail to determine a cost estimate.

c) Limited support for building reusable artifacts:

Agile processes such as Extreme Programming focus on building software products that solve a specific problem. Development in "Internet time" often precludes developing generalized solutions even when it is clear that this could yield long-term benefits. In such an environment, the development of generalized solutions and other forms of reusable software (e.g., design frameworks) is best tackled in projects that are primarily concerned with the development of reusable artifacts.

d) Limited support for development involving large teams:

Agile processes support process "management-in-the small" in that the coordination, control, and communication mechanisms used are applicable to small to medium sized teams. With larger teams, the number of communication lines that have to be maintained can reduce the effectiveness of practices such as informal face-to-face communications and review meetings. Large teams require less agile approaches to tackle issues

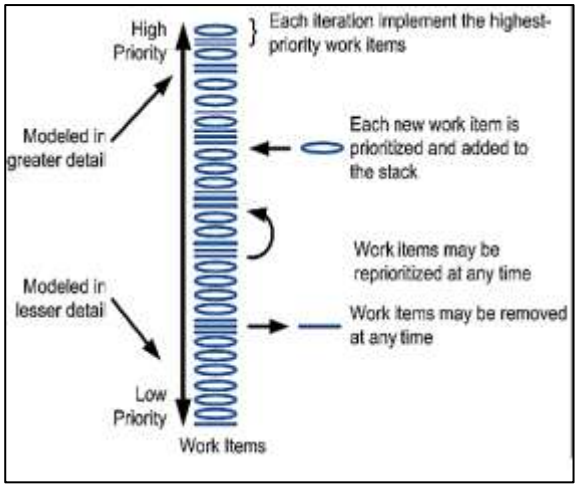

Figure 4: Process of Agile requirements change management

particular to "management-in-the-large". Traditional software engineering practices that emphasize documentation, change control and architecture-centric development are more applicable here.

e) Limited support for developing large, complex software:

The assumption that code refactoring removes the need to design for change may not hold for large complex systems in particular. In such software, there may be critical architectural aspects that are difficult to change because of the critical role they play in the core services offered by the system. In such cases, the cost of changing these aspects can be very high and therefore it pays to make extra efforts to anticipate such changes early. The reliance on code refactoring could also be problematic for such systems. The complexity and size of such software may make strict code refactoring costly and error-prone.

Traditional vs. Agile Software Development Methods

Throughout the literature we have consulted, a clear distinction between traditional and agile software development methods is made. My analysis found that a typical traditional method tells the developers pass through several phases in a prescriptive manner, phases conceived out of a plan based on the requirements specifications of the customer. Security in and stability of the project is therefore emphasized and extensive documentation is encouraged. An agile method on the other hand puts virtually all of the above in second place to complete flexibility and customer collaboration. Developers are agile as to being responsive to changing needs of the customers. Prototypes and beta versions are delivered constantly to the end users who in turn report back with changes that have to be made and wishes for additional functionality. The well-being of the development team also has a strong focus in agile software development. To illustrate the somewhat varying degree of agility in all discussed software development methods, an "*Axis of Agility*" is created where the methods relative placing are shown on a one-dimensional scale with the non-existent extremes of traditional and agile values at each end of the barometer. This Axis of Agility depicts subjective feelings for how each of the methods stands in relation to each other with regard to these values.

Finally, in order to summarize the characteristics of the traditional and the agile software development methods, we once again turn to the agile values which truly captures the essence of what agile software development methods, relative traditional software development methods, are all about.

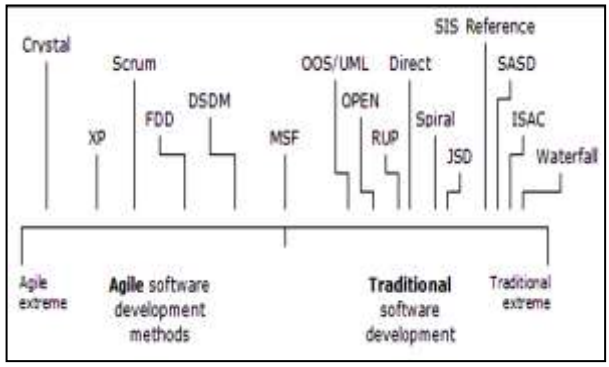

Figure 5: The Axis of Agility

Applicability of Agile software development methods

a) Where to use Agile Development

Agile Management & development methods are used under the following circumstances:

- The customers/users are active participants in requirements and/or analysis modeling efforts
- Changing requirements are welcomed and acted upon accordingly – there is no "requirements freeze"
- While working on the highest priority requirements first, as prioritized by our project stakeholders, and in turn focusing on highest risk issues as work progresses
- The content of models is recognized as being significantly more important than the format/representation of that content.

b) Where not to use Agile Development

Agile development methods aren't used under the following circumstances:

- When goal is to produce documentation, such as a requirements document, for sign-off by one or more project stakeholders

- While using a case tool to specify the architecture and/or design of our software BUT not using that specification to generate part or all of our software
- When customers/users have limited involvement with our efforts. For example they are involved with initial development of requirements, perhaps are available on a limited basis to answer questions, and at a later date will be involved in one or more acceptance reviews of our work

Conclusions

This paper aims for the identification of factors that help to decide for Agile or traditional project management methodologies in different types of projects. The following are the characteristics of the project which would benefit significantly if done with agile methodologies:

a) Poorly defined requirements
b) High level of complexity
c) Risk of failure is high
d) High development platform flexibility

Also several other factors were identified:

a) High risk tolerance of the project champion
b) High programmer experience with chosen development platform
c) High end-user ability to adopt to change
d) High budget elasticity

In comparison with Agile it was found that projects with high resource count and sub-contract development benefit more from traditional methodologies.

Theoretical Implications: - This paper aimed at finding out when does agile methodologies work best and when they do not work, hence it is better to do project with traditional methodologies. From the above it is possible to see that the following groups of attributes are influential for the selection Agile versus traditional methods: Requirements, project risk, team and development process characteristics. These groups include different factors within them which need to be accessed prior to selection of methodology.

Managerial Implications: - Main implication of this paper is that it provides a foundation for choosing methodology for the software projects. As it has been mentioned previously a choice of methodology may make or break the project. Project characteristics identified for the Agile project are not complete and there are many more which need to be taken into account. However if project does not have many of those characteristics it may be better to do it using traditional methodologies. Agile methods are quite complicated in comparison with traditional ones, thus for some projects they may be too excessive.

Future Research

The paper shows that agile software development methods requires a lot of skill and effort from those who use them. This may lead to that teams consisting of mainly inexperienced developers cannot fully reap the benefits of these methods thus leaving them dissatisfied with their choice of software development method. There may be reason for dissatisfaction among users of certain methods is that why a method may fail to satisfy a specific organization and succeed in another.

Another interesting aspect to be studied is , no or low customer participation would mean that Agile would not work at all, while low complexity would still mean that Agile could be used with this project.

References

1. Rashina Hoda, James Noble, Stuart Marshall: “The Impact of Inadequate Customer Collaboration on Self-Organizing Agile Teams”, 2010
2. Maarit Laanti, Outi Salo, Pekka Abrahamsson: “Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation”, 201
3. Subhas Chandra Misra , Vinod Kumar, Uma Kumar: “Identifying some important success factors in adopting agile software development practices”, 2009
4. Agile. (2009). "Cambridge Advanced learner's Dictionary Online." Retrieved 09/12/2009.
5. Helen Sharp, Hugh Robinson, Marian Petre: “The role of physical artefacts in agile software development: Two complementary perspectives”, 2009
6. Simona Motogna, I. Lazˇar, B. Pˆarv, I. Czibula: “An Agile MDA Approach for Service-Oriented Components”, 200
7. Ambler, S. (2008a). "Acceleration: An Agile Productivity Measure " Retrieved 08/12/2009.
8. Ambler, S. (2008b). "Results from Scott Ambler’s February 2008 Agile Adoption Survey."
9. Frank K.Y. Chan, James Y.L. Thong: “Acceptance of agile methodologies: A critical review and conceptual framework”, 2008
10. Woi Hin, Kee: “ Future implementation and integration of agile methods in software development and testing”, 2006
11. Beck, K., and Fowler, M. Planning Extreme Programming, Boston: Addison Wesley, 2001. Bossi, P., and Cirillo, F. “Repo Margining System: Applying XP in the Financial Industry,” in Proceedings of the 2nd International Conference on eXtreme Processing and Agile Processing Software Engineering (XP 2001), Villasimius, Italy, May 2001.
12. Cockburn, A. & Highsmith, J. 2001. Agile Software Development: The People Factor. Computer, Vol. 34, No. 11, pp. 131.133.
13. Highsmith, J. (2000). Adaptive Software Development: a Collaborative Approach to Managing Complex Systems, Dorset House Publishing Co., Inc.
14. Boehm, B. 1988. A Spiral Model of Software Development and Enhancement. Computer, Vol. 21, No. 5, pp. 61.72.
15. Boehm, B. (1981). Software Engineering Economics, Prentice-Hall.
16. Erickson, J. & Lyytinen, K. 2005. Agile Modelling, Agile Software Development, and Extreme Programming: The

State of Research. Journal of Database Management, Vol. 16, No. 4, pp. 88.100.
17. Highsmith, J. 2004. Agile Project Management, Creating innovative products. Addison-Wesley.
18. Cohen, D., Lindvall, M. & Costa, P. 2004. An Introduction to Agile Methods. Elsevier Academic Press. 0-12-012162 2-67.
19. Larman, C. (2004). Agile and Iterative Development:A Manager's Guide. C. Alistair and H. Jim, Pearson Education, Inc.
20. Larman, C. and V. R. Basili (2003). "Iterative and Incremental Development: A Brief History." IEEE Computer Society 36(6):
21. Cohn, M. & Ford, D. 2003. Introducing an Agile Process to an Organization. IEEE Computer, Vol. 36, No. 6, pp. 74.78.
22. Anderson, D. J. 2003. Agile Management for Software Engineering, Applying the Theory and Constraints for Business Results. Prentice Hall.