IJCSE International Journal of Computer Sciences & Engineering Open Access
Review Paper Volume-4, Special Issue-3, May 2016 E-ISSN: 2347-2693

Agile Software Development

Mr. Manjunath R[1] and Nagashree R A[2]

[1,2]*Dept. of Computer Science & Engineering, City Engineering College Bangalore, India*

Available online at: www.ijcseonline.org

Abstract— Successful software is one which provides quality product in given cost and time. Delivering quality software in definite time is a difficult task. Traditional software processes are heavy weight, giving importance to documentation and are rigid making them difficult to apply to different software projects. Agile has become one of the big buzzwords in the software development industry. To put it simply, Agile development or lightweight methods are less documentation oriented and more code oriented stating that source code is the most important document. Agile is a different way of executing software development teams and projects. Agile approaches help the teams respond to unpredictability through incremental, iterative work cadences or otherwise known as "sprints". Agile methodologies are an alternative to waterfall, or traditional sequential development. This software can be used in development stage, open collaboration and process adaptability in the process of project development. With a minimal work in different stages can improve planning of the project. . This paper discusses a few agile processes, the philosophy driving them and challenges faced while implementing them and mainly focuses on seeking alternative approach to traditional project management.

Keywords- Agile movement, software methodology, iterative tasks, light-weight methods

I. INTRODUCTION

In earlier days software development activity followed "code and fix" approach. This approach worked well for small systems but failed when systems grew larger and when there was need to add new features. To avoid this Engineering methodology came into picture. The aim of these methodologies was to make the software process more predictable and efficient by having a strong emphasis on planning activity. This approach works well for other engineering fields, like lot of planning is needed to build a bridge or a house. But the software market is ever changing and brings in greater choices into market. Users and managers must deal with issues like what to include and what to exclude in the software, which technologies to use, what will give the company a competitive edge?. These questions difficult to answer and trying to predict them in a rapidly changing market is even more difficult.

As a reaction to these methodologies, lightweight methodologies like agile methodologies appeared on the scene. Agile methodologies attempt to compromise between no process and too much process. According to [2] there are two more differences

- *Agile methods are adaptive rather than predictive.*

Engineering methods try to plan out in great detail for long span of time, this approach works well till there are no changes in design. Agile methods welcome change and try to adapt and thrive on change.

- *Agile methods are people oriented rather than process oriented.*

Engineering methods define processes so that they work no matter the skill of the workers. It says that individuals are not as important as their roles. This approach is correct for a factory where workers are not the most intelligent and creative people. Agile methods state that no process will ever make up for the skill and intelligence of the development team. So role of process is to support development team in their work.

Agile development is based on iterative incremental development, in which requirements and solutions evolve through team collaboration. It recommends a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a theoretical framework and does not specify any particular practice that a development team should follow. 'Scrum' is a specific agile process framework that defines the practices requires to be followed. Early implementations of agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear ,Extreme Programming (1996) , Adaptive Software Development , Feature Driven Development (1997) , and Dynamic Systems Development Method (DSDM) (1995). All these are collectively referred to as "Agile Methodologies", after the Agile Manifesto was published in 2001. The *Manifesto for Agile Software Development*, also known as the *Agile Manifesto*, was first proclaimed in 2001, after "agile methodology" was originally introduced in the late 1980s and early 1990s. The manifesto came out of the DSDM Consortium in 1994, although its roots go back to the mid-1980s at DuPont and texts by James Martin and James Kerr et al.

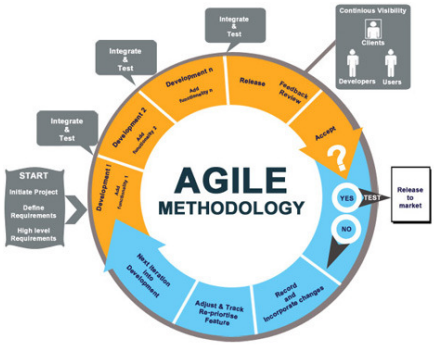

Fig:1

Agile principles

The Agile Manifesto is based on twelve principles:

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development
- Working software is delivered frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

2. AGILE PROCESS PHILOSOPHY

Agile software development philosophy has its roots in the reality of today's markets. Agile software processes attempt to deal with issues introduced by rapidly changing and unpredictable markets. The "Manifesto for Agile software development" [1] the basic ideas of the philosophy are introduced through four basic values:
i] Individuals and interactions over processes and tools.
ii] Working software over comprehensive documentation.
iii] Customer collaboration over contract negotiations.
iv] Responding to change over following a plan.

The items to the right have value, however, the items on the left define the agile philosophy. In this paper, we will focus on the left hand side items to explain the agile software development. We will now look at all the four aspects in detail.

i] Individuals and interactions

Using adaptive process requires a very effective team of developers. The team has to work well together to be more effective. Face to face meetings have special importance in agile processes. It is believed that people respond quicker and transfer ideas more rapidly when talking face to face than they can when reading or writing documentation.[3]. Extreme programming introduces the concept of pair programming where two developers develop a module together to provide much better and quicker output than the same job done individually. The concept of synergy (i.e. the interaction of two or more agents or forces so that the combined effect is greater than the sum of their individual effects) takes hold because a few designers, sharing a common space, working together, can produce more code quicker than can the same individuals working alone.[1].

In traditional methodologies treat people as resources that are like replaceable part. As stated earlier they say that individuals are not as important as their roles. They fail to understand that each individual is dynamic and unpredictable. When we are programming a computer, we are controlling a predictable device. But when handling human beings this approach fails. Treating individuals are replaceable resources reduces their morale and they look for much better working environments.

The traditional approach is more suited to a factory where the workers are not the most intelligent people or the best people to design and be creative. So here the approach of separating the people who plan the system and the people who construct is suited. But in software industry the developers are intelligent, capable and competent people so treating them in the same way does not help. According to [2] the Taylorist notion of a separate planning department that decides how things work works only if planners understand how to do the job than those doing it, if you have bright, motivated people doing the job then this does not hold true.

Another important aspect of individuals and interactions is that the team should be empowered to take all technical decisions. At times fast decision making is needed, if we have to wait every time for the management to approve it then it slows down the whole process of development. So power of taking technical decisions must rest in the hands of technical people

Reading [2] we are advised that team members and management must have an equal place in the project. This does not mean that technical people will take the role of management. Management will still retain its role of an

enabler but management should recognize the fact that development team can also take technical decisions.

ii] Working software

In agile processes source code is the single most important document while in the traditional approach (big design upfront) the requirement's document is the single most important document. In big design upfront (BDUF) it is possible to gather all the requirements upfront (beforehand), prior to writing any code. This is a classical approach which works well for mechanical industry where we gather all requirements, get the customer to agree on them and then apply procedures to restrict change. Gathering of requirements beforehand gives project a level of predictability. This predictability has a value and is very critical when the systems in consideration are Life critical systems where big requirements change could be a disaster. For all other systems this predictability adds a layer of documentation.

In this radically changing market it is not possible to gather complete set of customer requirements that are stable and unchanging. Customer is not always sure of what he wants. He gets a better understanding of it only when he sees a working model or a prototype of the system. This helps him to visualize the final system better. Essentially "In today's economy the fundamental business forces are changing the value of software features too rapidly. What might be a good set of requirements now, is not a good set in six months".[2].this means that as time passes the customer might want o add some feature which at that time looks essential to have.

According to [3] Agile processes link code and requirements tightly together. Users of agile processes view requirements as fluid and changing. XP (Extreme programming) introduces the idea of simplicity. It means not to add unnecessary artifacts or activities to a project. Eliminate everything not completely justified i.e. never produce documents predicting the future as they have a possibility of becoming outdated. According to [3] "the larger the amount of documentation becomes, more effort is needed to find the required information, and more effort to keep it up to date. For example as the system evolves or some changes are made to it, there arises a need to update the documentation that was made earlier. The first document the maintainer will go through to fix a bug is the source code rather than a pile of documents. So too much time should not be wasted on documentation, rather source code should be documented as good as possible. Making the code the actual design permits developers to move into the coding phase more rapidly.

iii] Customer Collaboration

Whenever software development is done by a separate firm the customers prefer fix price contracts in which they specify their requirements, ask for a quotation of price, and finally accept a price and leave the development to the firm. Agile processes require customer to be on site. They need customer to play an active part in the design process. Customer effectively is on the development team and works closely with developers to approve decisions and guiding the project through his perspective. This role is different from the traditional role of the customer and this change affects the business side of the project also.

Traditionally in a fixed price contract the development team generates a set of requirements leading to a predictable outcome, applying predictive processes to achieve goal. Agile methodologies say that requirements can never be stable so the fixed price development approach would not work in this case.

This does not mean we cannot budget what a project made using agile approach would cost. Agile approach is to fix time, price, and to allow the scope to vary in a controlled manner.[3]. As the customer has finer control on the project making changes based on feedback. At every iteration customer and development team can check the progress and decide with the development team whether to alter the direction of project. A different business model is required for such a setup. Hence it is essential that customer and supplier arrive at a business plan supporting customer collaboration over contract negotiations.

According to [2] a predictive process is often measured by how well it met its plan. A project on-time and on-cost is considered to be a success. For agile software development the question is business value-did the customer get software that's more valuable to them than the cost put in. Good agile software will build something different and better than the original plan foresaw.[2].

iv] Responding to change

Today's market is volatile and ever changing making it impossible for a predictive process to work on stable set of requirements. Responding to change than following the laid out plan is what makes agile software development successful in today's market. Software development is more of a design activity so it's hard to plan and price. For software development to be predictable there is need for plenty of time, a large team, and stable requirements which is not possible in small projects. The problem with predictive processes is their difficult to map new requirement to additional cost, as they cannot predict how much it would cost to implement the new requirement.

Software is intangible in nature, it is difficult to see what value a software feature has until we use it for real [2]. Only after seeing the early prototype or version can one understand what features are important what are not. So this means that requirements should be changeable. For example a new technology or standard comes up then the customer would demand compliance with it which is the need of the hour for the customer to use an up to date product.

So our approach should not be towards stopping change but to determine how to better handle inevitable changes in project. "External environmental changes cause critical variations. Because we cannot eliminate these changes, driving down the cost of responding to them is the only viable strategy".[3].

The idea to respond to change is using an iterative approach while developing where we produce working versions of final software frequently that handle subset of requirements. These working systems should be integrated in the end to produce the final system. This is better than having documents which can hide flaws. Untested code can also hide certain flaws. But when we have people working on the system we can unearth flaws faster.

Co-located teams working together producing code instead of high maintenance documentation can help increase productivity.[3]. These teams with onsite customer will produce code that better reflects the customer's requirements.

3. AGILE PROCESSES

There are many agile processes; in this paper we will discuss adaptive software development and Extreme programming.

A] Adaptive Software Development

ASD (adaptive software development) was developed by Jim Highsmith. It does not discuss milestones, methods, and deliverables. ASD gives importance on applying ideas originating in the world of complex adaptive systems. ASD provides fundamental base to develop adaptive systems from which arise agile and adaptive processes.[3]. Jim Highsmith states that the premise of ASD is that outcomes are naturally unpredictable, so we are wasting effort by planning for it. Planning in the world of changing requirements will never be successful. ASD replaces the evolutionary life cycle by adaptive life cycle as shown in figure 1.

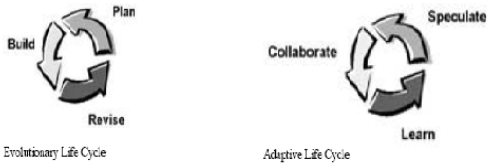

Figure 1.

Figure: Evolutionary Life cycle & Adaptive Life cycle [3].

ASD recognizes the fact that there is no point in experimenting endlessly in search of success. So the first phase of ASD is named "speculate" rather than planning which is not suitable for unpredictable world. Speculation means developing the good idea of where the project is heading, and put mechanisms in place to adapt to changing customer needs, changing technology and a changing market.

Collaboration replaces build because of ASD's recognition that people are essential while making a successful product. The customer collaborates in all activities of the software creation to get what he needs from the system. Collaboration is the activity of balancing: managing a project, such as configuration control and change management, with creativity the act of trusting people to find creative answers in an unpredictable environment.

Learning replaces revise because revise is backward looking.[3]. In the evolutionary life cycle revise means that while change is necessary it should be based on original plan i.e change cannot question original plan, it has to be in conformance with original plan. Learning is the act of gaining knowledge through experience. Learning is often discouraged in predictable environments; we may lay out things in advance and then follow then in design. In learning we can question all previous assumptions, using the results to decide in which direction to move.

ASD is not a methodology but rather is an approach that must be adopted by an organization when applying agile processes.

In an adaptive environment, learning challenges all stakeholders-developers and their customers- to examine their assumptions and to use the results of each development cycle to adapt to the next.

- Jim Highsmith

As learning is a continuous process designs and plans must change as development proceeds.

B] Extreme Programming

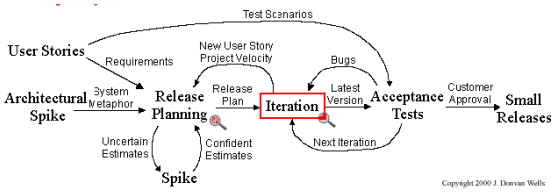

Figure: Extreme programming project [4].

The XP approach emphasizes customer involvement and testing:

Early on in the project, the team focuses on exploration and release planning; customer sits and writes stories, the programmers estimates them, the customer chooses the order in which the stories should be implemented.
Later, there is more focus on exploration. The team works iteratively; customer writes tests for the stories to be

successful and answers questions while programmers code.[5].

According to [2] the four basic building blocks (values) are

- Communication- Without communications project schedules slip, quality suffers, and the customer's wants and needs are misinterpreted or overlooked.
- Feedback- the need to check our results is important. Without feedback project might fail. Feedback tells how the project is doing while providing directions for future iterations. Feedback tells us where our project stands and what mistakes were made so that we don't make them again.
- Simplicity- Do not add unnecessary artifacts or activities to a project. Eliminate everything not completely justified.
- Courage- Putting faith in the people over the process requires courage. It is important to realize, however, that if processes do become more important than the people do, a project is headed toward failure.

The architecture of XP has following components

- Spike
- Metaphor
- First Iteration
- Small releases
- Refactoring
- Team practices

Spike

According to [5] during the release planning game, the team has the opportunity to do spikes: quick throw away (and thrown away) exploration into the nature of the solution.
We decide the system approach based on the stories and spikes. For example is the story is about managing orders on the Internet then the solution we might think of contains an application server, a web server, a database and a pair of firewalls. The spikes one does in the early phase guides us to the Deployment phase. Because spikes begin early on one can be prepared with the installation of hardware and software needed, so that project doesn't get halted because of inadequate resources.

Metaphor

An effective Metaphor helps guide your solution. In XP metaphor acts as a conceptual framework and provides a descriptive system of names. It identifies the key objects and their interactions. The metaphor may change as one's understand the system better.

First iteration

The first iteration is the key in making the system come together. From *Extreme Programming explained (*Kent Beck*)*
"The first iteration puts the system in place. Pick stories for the first iteration that will force one to create "the whole system" even if it is in skeletal form. "

Small releases

XP's small releases help jell the architecture quickly.[5].As we are installing a few months work, we are forced to get the essential structure together. We deliver the stories most important to the user first. So we get immediate feedback from the user which will help us correct the weak areas in the architecture.

Refactoring

Refactoring is improving a computer program by re-organizing its internal structure without altering its external behavior. It helps us manage design without changing the system's behavior; therefore, we don't risk the functionality of our program while we improve its architecture.[5].

Team Practices

The software architecture document is useful only if it tells how developers implement the things the system is supposed to do. XP forges the Software Architecture Document that RUP (Rational unified process) values, but still has architecture. Pair programming helps ensure that the people know and use the approach the team is using.[5].

Implementing Agile Processes

It is difficult to introduce agile processes into an organization because of the resistance offered by the employees to change in organizational structure. Adopting agile processes will change job profiles and roles and pay structure radically, so there is resistance. Developers are either over enthusiastic about it or highly skeptical about it. The over enthusiastic developers feel that agile means "moving quickly" meaning minimum discipline. It is important to understand that in agile processes decisions are taken with forethought and reason and it's not experimentation. Other developers resist it because they are followers of traditional approach and are familiar with its working and believe that agile processes can produce quality products.

Management is also uncomfortable with agile processes as they cannot use Gantt charts and other documents to manage the project.[3]. Traditionally managers analyze the progress by seeing which artifacts have been created. More emphasis on code and less on documentation worry them.

4. BENEFITS OF AGILE SOFTWARE DEVELOPMENT

Agile methods grew out of the real-life project experiences of leading software professionals who had experienced the challenges and limitations of traditional waterfall development on project after project. The approach promoted by agile development is in direct response to the issue associated with traditional software development both in terms of overall philosophy as well as specific processes.

Agile development, in its simplest form, offers a lightweight framework for helping teams, given a constantly evolving functional and technical landscape, maintain a focus on the rapid delivery of business value (i.e., œbang for the buck). As a result of this focus, the benefits of agile software development are that organizations are capable of significantly reducing the overall risk associated with software development. Customer satisfaction by rapid, continuous delivery of useful software. Working software is delivered frequently (weeks rather than months). Even late changes in requirements are also welcomed.

In particular, agile development accelerates the delivery of initial business value, and through a process of continuous planning and feedback, is able to ensure that value is continuing to be maximized throughout the development process. As a result of this iterative planning and feedback loop, teams are able to continuously align the delivered software with desired business needs, easily adapting to changing requirements throughout the process. By measuring and evaluating status based on the undeniable truth of working, testing software, much more accurate visibility into the actual progress of projects is available. Finally, as a result of following an agile process, at the conclusion of a project is a software system that much better addresses the business and customer needs.

5. DISADVANTAGES WITH AGILE

In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

There is lack of emphasis on necessary designing and documentation.

The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

5. CONCLUSION

Large organizations have heavy investments done in BDUF (big design up front) processes. They have preference for BDUF than agile processes as they have business models requiring fixed price contracts and strong inclination towards software engineering concepts. Life critical systems project have the severity to use BDUF processes, which also increases their cost and time to produce. Organizations may have to learn to use agile processes for their survival in market and be competitive.

We summarize that following factors suggest an adaptive process

- Uncertain or volatile requirements
- Responsible and motivated developers
- Customers who understand and are willing to get involved.

These factors suggest a predictable process

- A team of hundred
- Fixed price, fixed scope and contract.

One of the biggest challenges facing is how they handle larger systems.[2].Extreme programming explicitly says that it has been created for teams around 20 people. The message is one should go agile when the requirements are not stable and one cannot have a stable design and follow a planned process.

References

[1] Manifesto for Agile software development; http://agilealliance.com
[2] New methodology; Fowler; Martin; http://www.martinfowler.com/articles/newMethodology.html
[3] Agile Software Development Processes- A Different approach to Software design; Keith, Everette R; http://www.agilealliance.com/articles/articles/ADifferentApproach.pdf
[4] www.extremeprogramming.org; Last modified January 26, 2003;
[5] Extreme Programming Explored; Wake, William; Addison Wesley ISBN 0-201-73397-8; July 2001; Chapter 5