Globus An International Journal of Management & IT
A Refereed Research Journal
Vol 7 / No1 / Jul-Dec 2015 ISSN: 0975-721X

A STUDY ON EXISTING AGILE METHODS

* Shamti Sarkhel

Introduction

Agile – devoting "the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion" as mentioned in the Oxford Dictionary – software development methods are attempting to offer once again an answer to the eager business community asking for lighter weight along with faster and nimbler software development processes. To name a few of those developed: Adaptive Software Development (ASD), Agile Modeling, Crystal Methods, Dynamic System Development, Lean Development and Scrum. All these methodologies acknowledged that high quality software and more importantly customer satisfaction could only be achieved by bringing "lightness" to their processes. Some of the most used agile methodologies are listed below.

Extreme Programming (XP)

Extreme programming (XP) has evolved from the problems caused by the long development cycles of traditional development models. The XP process can be characterized by short development cycles, incremental planning, continuous feedback, reliance on communication, and evolutionary design. With all the above qualities, XP programmers respond to changing environment with much more courage. Further according to Williams, XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes on team building many times each day. The term 'extreme' comes from taking these commonsense principles and practices to extreme levels.

Scrum

Scrum is an iterative, incremental process for developing any product or managing any work. Scrum concentrates on how the team members should function in order to produce the system flexibility in a constantly changing environment. At the end of every iteration it produces a potential set of functionality. The term 'scrum' originated from a strategy in the game of rugby where it denotes "getting an out-of-play ball back into the game" with teamwork.

Scrum does not require or provide any specific software development methods/practices to be used. Instead, it requires certain management practices and tools in different phases of Scrum to avoid the chaos by unpredictability and complexity.

Feature Driven Development (FDD)

Feature Driven Development (FDD) was used for the first time in the development of a large and complex banking application project in the late 90's. Unlike the other methodologies, the FDD approach does not cover the entire software development process but rather focuses on the design and building phases.

The first three phases are done at the beginning of the project. The last two phases are the iterative part of the process which supports the agile development with quick adaptations to late changes in requirements and business needs. The FDD approach includes frequent and tangible deliverables, along with accurate monitoring of the progress of the report.

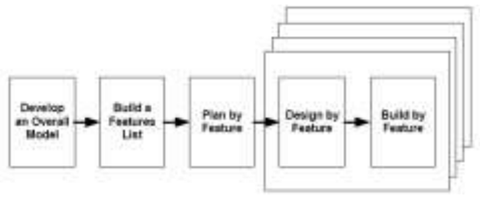

Figure : Feature Driven Development Process

- Develop an Overall Model - A high level walkthrough of the system scope and its context is performed by the *domain expert* to the team members and *chief architect*. Documented requirements such as use cases or functional specifications are developed.
- Build a Features List - A categorized list of features to support the requirements is produced.
- Plan by Feature - The development team orders the feature sets according to their priority and dependencies and assigned to c*hief programmers*. Furthermore, the classes identified in the first phase are assigned to c*lass owners (individual developers)*. Also schedule and milestones are set for the feature sets.
- Design by Feature & Build by Feature - Features are selected from the feature set and feature teams needed to develop these features are chosen by the class owners. The design by feature and build by feature are iterative procedures during which the team produces the sequence diagrams for the assigned features. These diagrams are passed on to the developers who implement the items necessary to support the design for a particular feature. There can be multiple feature teams concurrently designing and building their own set of features. The code

*MPhil Student, Swami Vivekanand University, Sagar, MP

developed is then unit tested and inspected. After a successful iteration, the completed features are promoted to the main build.

Dynamic System Development Method (DSDM)

The DSDM, Dynamic System Development Method, was developed in the United Kingdom in the mid-1990. It is a blend of, and extension to, rapid application development and Iterative development practices. Martin Fowler, one of the writers of Agile Manifesto, believes, “DSDM is notable for having much of the infrastructure of more mature traditional methodologies, while following the principles of the agile methods approach”. The fundamental idea behind DSDM is to fix time and resources, and then adjust the amount of functionality accordingly rather than fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality

Adaptive Software Development (ASD)

Adaptive Software Development (ASD), developed by James A. High smith, offers an agile and adaptive approach to high-speed and high-change software projects. It is not possible to plan successfully in a fast moving and unpredictable business environment. In ASD, the static plan-design life cycle is replaced by a dynamic speculate-collaborate-learn life cycle.

ASD focal point is on three non-linear and overlapping phases :

- Speculate - To define the project mission, make clear the realization about what is unclear.
- Collaborate – Highlights the importance of teamwork for developing high-change systems
- Learn – This phase stresses the need to admit and react to mistakes, and that requirements may well change during development.

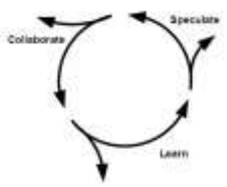

Figure : ASD Lifecycle

Since outcomes are naturally unpredictable, high smith views planning as a paradox in an adaptive environment. Normally in traditional planning when things do not go to plan it is seen as a mistake that should be corrected. However in an adaptive environment deviations guide us towards the correct solution.

ASD focuses more on results and their quality than the tasks or the process used for producing the results. In an unpredictable environment you need people to collaborate in a certain manner to deal with the uncertainty. Management is more about encouraging communication rather than telling people what to do, so that more creative answers are delivered.

In traditional predictive environments, designs are followed the same way they were laid out, therefore learning is discouraged. High smith points out, “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 the next”. As such learning is a continuous and important feature, one that assumes that plans and designs must change as development proceeds.

ASD does not have detailed principles like XP, but rather it provides a framework on how to encourage collaboration and learning within the project. ASD is not presented as a methodology for doing software projects but rather it is an approach or an attitude that must be adopted by an organization when applying agile processes.

References

1. Fitzgerald, B., Hartnett, G. and Conboy, K., “Customising Agile Methods to Software Practices at Intel Shannon”, European Journal of Information Systems, 15, 200-213, 2006.
2. G. Benefield, “Rolling out agile in a large enterprise,” in 41st Hawaii International Conference on System Science. 2008, IEEE Computer Society.
3. Jayakanth Srinivasan and Kristina Lundqvist, “Using agile methods in software product development: A case study,” Information Technology: New Generations, Third International Conference, pp. 1415–1420, 2009.
4. John Hunt, Agile Software Construction, Springer, September 2005.
5. K. Beck, Extreme Programming Explained: Embrace Change, second ed., Addison-Wesley, 2004, ISBN 978-0321278654.
6. K. Beck, Extreme Programming Explained: Embrace Change, Addison- Wesley, 2000, ISBN 0-201-61641-6.
7. Lyard, A. and Orci, T. (2000), Dynamic CMM for Small organisations. Proceedings ASSE 2000, the first Argentine Symposium on Software Engineering, September 2000, Tandil- Argentina, pp.133-149.