IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727Volume 7, Issue 3 (Nov. - Dec. 2012), PP 20-23
www.iosrjournals.org

Where to use agile methodology in Software development and its different models

Shamsher Alam
Bangalore

Abstract: *Agile methods were developed to overcome shortcomings and weaknesses in conventional software engineering. It can provide important benefits, but it is not applicable to all projects.*

In today economy, it is always difficult and impossible to predict how a computer-based system (e.g. a web based application) will evolve as time passes. It is really difficult to define or fix requirements fully before the project begins. So our main aim is to explore all the ways by which we can use agile method in software development to execute the project successfully within time.

An agile philosophy for software engineering stresses four key issues: the importance of self-organizing teams that have control over the work they perform; communication and collaboration between team members and between practitioners and their customers; a recognition that change represents an opportunity; and an emphasis on rapid delivery of software that satisfies the customer. Agile process models have been designed to address each of these issues.

We will find out which method is best suited for software projects using agile methodology. We will discuss all the methods which agile supports.

At the end we will be able to conclude that where agile fits best and which type of project can be done using this method.

Key Words: *XP- Xtreme Programming. CRC – Class-responsibility collaboration*

I. Introduction

An agile process can be applied to any software process. But it is best suited for the project where the requirement is not fixed before the start of the project. It combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of software, small, highly motivated project teams, informal methods, minimal software engineering work products and overall development simplicity. The development guidelines stress delivery over analysis and design and active continuous communication between developers and customers.

Software engineers and other project stakeholders (managers, customers, end-users) work together on an agile team – a team that is self- organizing and in control of its own destiny. An agile team fosters communication and collaboration among all who serve on it. The modern business environment that spawns computer-based systems and software products is fast-paced and ever-changing. Agile software engineering represents a reasonable alternative to conventional engineering for certain classes of software and certain types of software projects. It has been demonstrated to deliver successful system quickly.

Agile development might best be termed "software engineering lite". The basic framework activities-customer communication, planning, modeling, construction, delivery and evaluation - remain. But they morph into a minimal task set that pushes the project team toward construction and delivery (some would argue that this is done at the expense of problem analysis and solution design.

Customers and software engineers who have adopted the agile philosophy have the same view- the only really important work product is an operational "software increment" that is delivered to the customer on the appropriate commitment date.

Agility has become today's buzzword when describing a modern software process. Every-one is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Change in the software being built, change to the team members, change because of new technology, and change of all kind that may have an impact on the product they build or the project that creates the product.

II. Materials and Methods

What is an Agile Process?

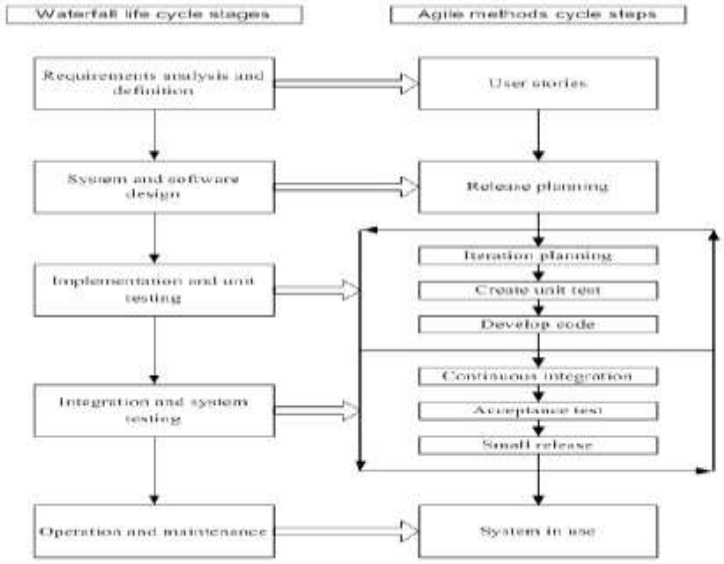
Any agile software process is characterized in a manner that addresses three main assumptions about the majority of software projects:

1. It is hard to predict in advance which requirements will persist and which will change. It is also difficult to predict how customer priorities will change as project progresses.

2. For most of the software, design and construction are interleaved. That is both the activities should e performed in tandem.
3. Analysis, design, construction, and testing are not as predictable (from a planning point of view).

The Agile Manifesto:
Individuals and interactions over process and tools.
Working software over comprehensive documentation.
Customer collaboration over contract negotiation.
Responding to change over following a plan.

Comparison of Conventional Process Model and Agile Model :



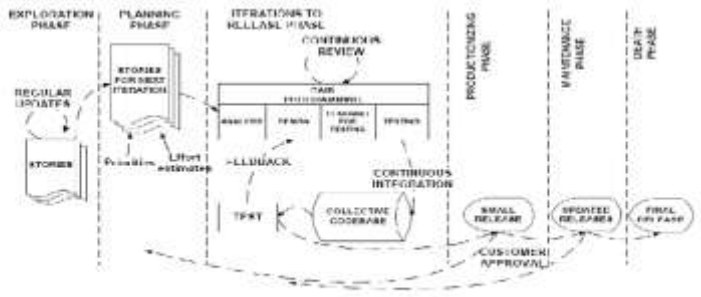
Agile Process Models:

We will present an overview of a number of different agile process models. There are many similarities (in philosophy and practice) among these approaches. Our intent will be to emphasize those characteristics of each method that make it unique. It is important to note that all agile models conform (to a greater or lesser degree) to the manifesto for agile software development and the principles.
All the below are the different models of agile method.

I. Extreme Programming (XP)
II. Adaptive Software Development (ASD)
III. Dynamic Systems Development Method (DSDM)
IV. Scrum
V. Crystal
VI. Feature Driven Development (FDD)
VII. Agile Modeling (AM)

Extreme Programming: This method was developed during early 1980s by Kent Beck. It uses an object-oriented approach as its preferred development paradigm. It encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

Planning: The planning activity starts with the creation of a set of stories (also called user stories) that describes required features and functionality for software to be built. Each story is written by the customer and is placed on an index card. The customer assigns a value (i.e., a priority) to the story based on the overall business value of the feature or functionality. Members of the XP team then assess each story and assign a cost-measure in development weeks-to it. If the story will require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.



Life cycle of XP Process

Design: XP design rigorously follows the KIS (Keep it simple) principle. A simple design is always preferred over a complex representation. In addition, the design provides implementation guidance for a story as it is written- nothing less, nothing more.
XP encourages the use of CRC cards as an effective mechanism for thinking about the software in an object-oriented context.

Coding: XP recommends that after stories are developed and preliminary design work is done, the team should not move to code, but rather develop a series of unit tests that will exercise each of the stories that is to be included in the current release. Once the unit test has been crated, the developer is better able to focus on what must e implemented to pass the unit test.
A key concept during the coding activity is pair programming. XP recommends that two people work together at one computer workstation to create code for a story. This provides a mechanism for real time problem solving (two heads are always better than one) and real time quality assurance. It also keeps the developers focused on the problem at hand.
Testing: We have already noted that the creation of a unit test befor coding commences is a key of the XP approach. The unit tests that are created should be implemented using a framework that enables them to be automated (hence, they can be executed easily and repeatedly). This encourages a regression testing strategy whenever code is modified.

XP acceptance tests, also called customer tests, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer. Acceptance tests are derived from the user stories that have been implemented as part of a software release.

Adaptive Software Development (ASD) : Adaptive software is used as a technique for building complex software systems. It incorporates three phases :
1) Speculation
2) Collaboration
3) Learning

Speculation: During speculation, the project is initiated and adaptive cycler planning is conducted. Adaptive cycle planning uses project initiation information – the customer's mission statement, project constraints (e.g delivery dates or user descriptions) and basic requirements to define the set of release cycles that will be required for the project.

Collaboration: Motivated people work together in a way that multiplies their talent and creative output beyond their absolute numbers. This collaborative approach is a recurring theme in all agile methods. But collaboration is not easy. It is not simply communication, although communication is a part of it.
People working together must trust one another to (1) criticize without animosity; (2) assist without resentment; (3) work as hard or harder as they do; (4) have the skill set to contribute to the work at hand.

Learning: As members of an ASD team beging to develop the components that are part of an adaptive cycle, the emphasis is on learning as much as it is on progress toward a completed cycle. ASD team learn in three ways :
1) **Focus groups** : The customer and/or end-users provide feedback on software increments that are being developed. This provides a direct indication of whether or not the product is satisfying business needs.
2) **Formal technical reviews :** ASD team members review the software components that are developed, improving quality and learning as they proceeds.

3) **Postmortems:** The ASD team becomes introspective, addressing its own performance and process.

Dynamic Systems Development Method (DSDM):

The Dynamic Systems Development Method (DSDM) is an agile software development approach that provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment.

DSDM can be combined with XP to provide a combination approach that defines a solid process model with the nuts and bolts practice (XP) that are required to build software increments. In addition, the ASD concepts of collaboration and self-organizing teams can be adapted to a combined model.

Scrum:

Scrum (the name derives from an activity that occurs during a rugby match) is an agile process model that was developed by Jeff Sutherland and his team in the early 1990s. In recent years, further development of the scrum methods has been performed by Schwaber and Beedle. Scrum principles are consistent with the agile manifesto:

- Small working teams are organized to maximize communication, minimize overhead, and maximize sharing of tacit, informal knowledge.
- The process must be adaptable to both technical and business changes to ensure the best possible product is produced.
- The process yields frequent software increments that can be inspected, adjusted, tested, documented and built on.

Crystal:

Alistair Cockburn and Jim Highsmith created the crystal family of agile methods in order to achieve a software development approach that puts a premium on maneuverability during the Cockburn characterizes as a resource-limited, cooperative game of invention and communication, with a primary goal of delivering useful, working software and a secondary goal of setting up for the next game.

Feature Driven Development (FDD):

In the context of FDD, a feature is a client-valued function that can e implemented in two weeks or less. The emphasis on the definition of features provide the following benefits :

- Because features are small blocks of deliverable functionality, users can describe them more easily, understand how they relate to one another more readily and better review them for ambiguity, error, or omission.
- Features can be organized into a hierarchical business related grouping.
- Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.
- Because features are small, their design and code representations are easier to inspect effectively.
- Project planning, scheduling, and tracking are driven by the feature hierarchy, rather than an arbitrarily adopted software engineering task.

References:

[1] Advanced Development Methods, Inc. "Origins of Scrum"
[2] The agile Alliance Home Page, http://www.agilealliance.org/home
[3] Cockburn, A., and Agile Software Devlopment, Addision-Wesley 2002.
[4] Software Engineering – Roger S . Pressman