International Journal For Technological Research In Engineering
Volume 1, Issue 7, March-2014 *ISSN (Online): 2347 - 4718*

A SURVEY ON LIGHTWEIGHT METHODOLOGY USED FOR SOFTWARE DEVELOPMENT

Pramod Saxena[1], Dr. Manju Kaushik[2], Suyash Raizada[3]
[1]Department of M.tech (SE), [2]Associate Professor (CSE), [3]M.tech (SE)
JECRC University
Jaipur, India.

Abstract: **Lightweight methodology that utilizes iterative development and prototyping are widely used in variety of software industry projects which can satisfy to the changes of requirements. Little iterations are used that are required for efficient product delivery. Traditional software development processes are not much efficient to manage the rapid change in requirements. Despite the advantages of Lightweight methodology's a state that it fails to pay attention to architectural and design issues and therefore is bound to produce small design-decisions. Here, in this paper we identify the impacts that Lightweight methodology has on software development processes with respect to quality within the organizational, methodical, and cultural framework.**

Keywords: **Agile Alliance, Agile Methodology, Crystal Method, Extreme Programming, Feature Driven Development, Scrum, Test Driven Development.**

I. INTRODUCTION

In typical software development process it is assumed that all the requirements are complete and can be implemented directly in order to develop the application, but this is not the case for most of the projects today. In modern competitive era changes are frequent to any software product or module which is under development, due to the market competitions priority of requirements changes frequently and only specific development is done which is urgently required and then later on changes and improvements comes into the picture for the rest developed modules. So requirement engineering is done in parallel to software development and requirement changes often happen to survive in the competitive market.

Whenever a new requirement comes into the picture it takes lot of effort in terms of time and cost for analysis and implementation. Theoretically change requirements takes less time than typical development requirements but practically it takes almost the same or even more time as development for complex change requirements. Since changes of any type whether simple or complex needs a complete software development lifecycle, because after analyzing the requirement it is implemented and integrated with the existing code and then implemented requirement is verified against the test cases and also verified against the functionality required.

Once implementation is done and verified, lot of refactoring related work is required for making sure that the implemented code is written in standard format and integrated with the system as per the development standards. Refactoring is also an important type of change requirement which sticks the development policies with the developed code and which comes into the picture once the development task in bulk is over. Refactoring improves code, usually increasing the function while reducing code bulk. However, such refactoring or restructuring often forces the application to undergo a complete development cycle, including unit, acceptance, and regression testing, followed by subsequent redeployment. In a large IT or engineering production system, this can be time consuming and error prone.

Lightweight methodology is design for change, without refactoring and rebuilding. Its objective is to design programs that are receptive to change. Ideally, Lightweight methodology lets changes be applied in a simple, localized way to avoid or substantially reduce major refactoring, retesting, and system builds.

Lightweight Methodologies are a group of software development methods that are based on iterative and incremental development. The four major characteristics that are fundamental to all lightweight methodologies are: adaptive planning, iterative & evolutionary development, rapid and flexible response to change and promote communication [1, 2]. Its main emphasis is in obeying the principles of "Light but sufficient" and being people-oriented and communication-centered. As it is named as lightweight process, it is more suitable for the development of small projects [3]. Now we have focus of Agile software development they were takes the view that production teams should start with simple and predictable approximations to the final requirement and then continue to increment the detail of these requirements throughout the life of the development. This incremental requirements refinement further refines the design, coding and testing at all stages of production activity. In this way, the requirements work product is as accurate and useful as the final software itself [4].

The principle of agile software development proposes [5] that "at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly". In other terms it may be said that agile methodology addresses exactly the challenges of an unpredictable, disordered business and technology environment [6]. Lightweight methods that include Scrum, Crystal Clear, Extreme Programming (XP), Adaptive Software Development (ASD), Feature Driven Development (FDD), and Dynamic Systems Development Method (DSDM) Crystal, Lean Software Development etc. [7]. Agile methods break tasks into small increments with minimal planning called Iterations. Iterations are short time frames that runs from one to four weeks. Each iteration involves a team working through a full software development cycle, including planning, requirements analysis, design, coding, unit testing, and acceptance testing. This minimizes overall risk and allows the project to adapt to changes quickly. Most of the agile implementations use a formal daily face-to-face communication among team members. In this brief communication, team members report to each other what they did the previous day, what they intend to do today, and what are the hurdles they faced When customer or domain expert works directly with the development team everyone learns something new about the problem [9, 10,11].

II. POPULAR LIGHTWEIGHT METHODOLOGIES

Some of the most commonly used methodologies are discussed in this section. There are several parameters associated with the choice of these techniques, some of them are team size, iteration length and support for distributed environment. These parameters are also discussed for these most commonly used techniques here:

A. Extreme Programming (XP)

Extreme programming is a good agile methodology when the team size is generally small i.e. from 2 to 10. Iteration length is generally short around 2 weeks. XP is not suitable for distributed teams. The goal of Extreme programming (XP) is to improve software quality and responsiveness to changing customer requirements.
Advantages: Lightweight methods suit small medium size projects. Produces good team cohesion and emphasizes final product and Iterative. Test based approach to requirements and quality assurance.
Disadvantages: Difficult to scale up to large projects where Documentation is essential and needs experience and skill if not to degenerate into code-and-fix. Programming pairs is costly.

B. Scrum

A SCRUM is a Rugby team of eight individuals [15]. The team acts together as a pack to move the ball down the field. Teams work as tight, integrated units with a single goal in mind. In a similar manner, the SCRUM software development process facilitates a team focus. SCRUM is a light SDLC methodology for small teams to incrementally build software in complex environments. SCRUM is most appropriate for projects where requirements cannot be easily defined up front and chaotic conditions are anticipated. SCRUM divides a project into sprints (iterations) of 30 days. Functionality is defined before a sprint begins. The goal of the process is to stabilize requirements during a sprint.
Advantages: High amount of risk analysis. Good for large and mission-critical projects. Software is produced early in the Software life cycle.
Disadvantages: Can be a costly model to use. Risk analysis requires highly specific expertise. Project’s success is highly dependent on the risk analysis phase. It doesn’t work well for smaller projects.

C. Feature Driven Development (FDD)

Feature Driven Development (FDD) is a model-driven short-iteration software development process. The FDD process starts by establishing an overall model shape. This is followed by a series of two-week “design by feature, build by feature” iterations. FDD consists of five processes: develop an overall model, build a features list, plan by feature, and design by feature, and build by feature. There are two types of developers on FDD projects: chief programmers and class owners. The chief programmers are the most experienced developers and act as coordinator, lead designer, and mentor. The class owners do the coding. One benefit of the simplicity of the FDD process is the easy introduction of new staff. FDD shortens learning curves and reduces the time it takes to become efficient. Finally, the FDD methodology produces frequent and tangible results. The method uses small blocks of user-valued functionality. In addition, FDD includes planning strategies and provides precision progress tracking.

D. Crystal Method

Crystal methods are based on the principle that how to achieve a maximum extent by which a written communication or documents communication can be reduced to a verbal communication for faster development. All Crystal methods begin with a core set of roles, work products, techniques, and notations. There is no limit on team size in crystal methods. Iteration lengths are large generally 4 months and more. It is built to support distributed team. These techniques are based on the following four principles:

a) Use larger methodologies for larger teams.
b) Use denser methodologies for more critical projects
c) Interactive, face-to-face communication is most effective.
d) Weight is costly.

E. Wisdom

The White-water Interactive System Development with Object Models [16] addresses the needs of small development teams who are required to build and maintain the highest quality interactive systems. The Wisdom methodology has three key components: A software process based on user-centered, evolutionary, and rapid prototyping model. A set of conceptual modeling notations that support the modeling of functional and nonfunctional components. A project management philosophy based on tool usage standards and open documentation. Wisdom is comprised of three major workflows: requirements workflow, analysis workflow, and

design workflow. In addition, the methodology is based on seven models and uses four types of diagrams.
Task flow plays an important role in Wisdom and corresponds to a technology-free and implementation independent portrayal of user intent and system responsibilities.

F. DSDM

The Dynamic Systems Development Method (DSDM) is a framework [8] used to control software development projects with short timelines. It was developed in 1994 by a consortium formed by a group companies in Great Britain. The methodology begins with a feasibility study and business study to determine if DSDM is appropriate. The rest of the process consists of three interwoven cycles. These are functional model iteration, design and build iteration, and implementation. The underlying principles of DSDM include frequent deliveries, active user communication, empowered development teams, and testing in all phases of a project. DSDM is different than traditional approaches in that requirements are not fixed. Project requirements are allowed to change based upon a fixed timeline and fixed project resources. This approach requires a clear prioritization of functional requirements. Emphasis is also put on high quality and adapting to changing requirements. It has the advantage of a solid infrastructure (similar to traditional methodologies), while following the principles of lightweight SDLC methods.

G. ASD

It is known as Adaptive Software Development (ASD). It means we are work on adaptive nature of SDLC methodologies. They were inherently flawed, in modern business processes. Adaptive Software Development (ASD) as a framework from which to address the rapid pace of many software projects [18]. ASD is grounded in the science of complex adaptive systems theory and has three interwoven components: the Adaptive Conceptual Model, the Adaptive Development Model, and the Adaptive
Management model. In contrast to the typical waterfall (plan, build, implement) or the iterative (plan, build, revise) life cycles, the adaptive development life cycle (speculate, collaborate, learn) acknowledges the existence of uncertainty, change and does not attempt to manage software development using precise prediction and rigid control strategies. ASD is grounded in the science of complex adaptive systems theory and has three interwoven components: the Adaptive Conceptual Model, the Adaptive Development Model, and the Adaptive (leadership-collaboration) Management Model. These process are work as applications are a closer match to customer requirements due to constant evolution and business needs. Than they were development process adapts to specified quality. Finally they were reduced risk and established the project.

H. ASP

With the rapid change in the requirements in terms of budget, schedule, resources, team and technology agile model responds to changes quickly and efficiently. Agile is an answer to the eager business community asking for lighter weight along with faster and nimbler software development processes [19].
Following are the main principles to implement an agile model:

a) Agile team and customer must communicate through face-to-face interaction rather than through documentation.
b) Agile team and customer must work together throughout the development.
c) Supply developers with the resources they need and then trust them to do their jobs well.
d) Agile team must concentrates on responding to change rather than on creating a plan and then following it.
e) Emphasis on good design to improve quality.
f) Agile team must prefer to invest time in producing working software rather than in producing comprehensive documentation.
g) Satisfy the customer by “early and continuous delivery of valuable software”.

III. LITERATURE REVIEW

Various studies and surveys have been made that shows Lightweight methodology’s popularity based on characteristic of requirements, small or big organization, and experience of the project team. Lightweight methods have proven their effectiveness and are transforming the software industry. A high percentage of software development efforts have no process and might best be described as a chaotic “code and fix” activity. Light SDLC techniques are a compromise between no process and too much process. In the following sections, literature related to nine types of lightweight SDLC methodologies is discussed. Some of the A Survey on Lightweight methodology are presented here.
As demonstrated by Andrew et al [1] using a survey based approach, agile methodology is favorable due to improved communication between team members, quick releases and flexibility of designs. Scrum methodology is the most popular; and test driven development and pair programming are the least used practices.
As demonstrated by A. Ahmed et al [9], scrum is used most commonly, 50% of the projects are done with active stakeholder participation. 66.7% participants were agreeing that productivity has improved and quality is improved by 50%.
Based on a study, Pirjo et al [8] shows that agile methods are good for some programming environments, but not for all. Projects that involve large teams, well-defined requirements, clients needing high assurance and large code-bases, the traditional plan-oriented project profile works well. Therefore, Agile methods produces best results in case of when the team is small, the requirements are not yet well defined, the project code base is small and the customer is

interested in seeing significant progress. However, as a software project transitions from a small prototype to a large stable system with a large team, with promises to keep and dates to meet, then agile methods alone is not sufficient then some additional mechanism is needed.

Tore et al [14] report that XP is seemed difficult to introduce in large, complex organizations but easier in other organization types. Pair programming is inefficient and XP works best with experienced development teams. Also there is lack of attention to design and architectural issues.

Behrouz Far [12] mapped the software reliability engineering into an agile development process. As per the study, test driven development seems to be incompatible with the reliability model.

Markus et al [13] highlights the negative impact of change in requirements on customer satisfaction. The main contribution of their paper pertains to the interaction effects between change in requirements and agile methods on customer satisfaction. They found that work climate, final product adaptability and willingness to adapt to change have a positive moderating effect on the relationship between change in requirements and customer satisfaction.

The study performed by Sharifah Syed et al [14] shows that agile methodology is more people-oriented than process oriented in a more volatile environment. Excepting the satisfaction of the developers this is helpful only when the requirements are uncertain or volatile.

IV. BENEFITS OF LIGHTWEIGHT METHODOLOGY IN SOFTWARE DEVELOPMENT

The key benefits of lightweight methodology in software development due to which lightweight (agile) methodology should be adopted while developing software are shown in the figure1 and explained in detail thereafter.

1. Requirements Changes: Planning phase is dramatically improved. First, because customers are directly involved in the development process, that is, customers control the processes of projects through on-site interaction, requirements truly reflect the current needs of the end users.
2. Testing and Problem Detection: As testing is performed during each iteration, faults are detected earlier and can be fixed before it increases in severity than with a plan-driven process model. Also, continuous testing allows continuous testing feedback, which further improves code developed in future iterations.
3. Increased Performance: Daily standup meetings provide an opportunity to exchange valuable information and to fine tune improvements continuously. The ability to discuss complex projects through simple stories and simple design encourages teamwork. Better communication leads to increased knowledge sharing, self-organizing teams, and team morale as employees begin to trust and gain the trust of their team members. This increases team productivity and generates better performance in terms of good Return on Investment than the sum of all individual output.

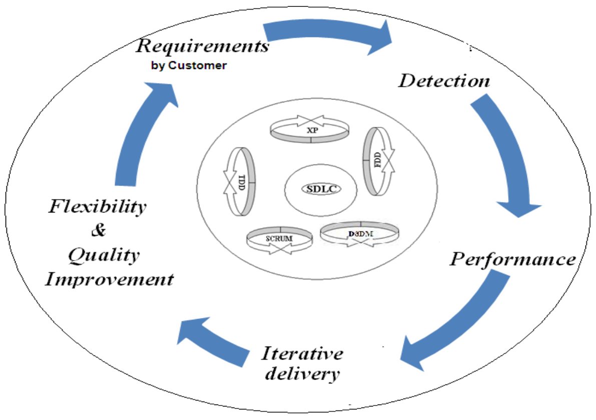

Fig.1. Software Development Methodologies with Benefits

4. Iterative and incremental delivery: Project delivery is divided into small functional releases or increments to manage risk and to get early feedback from customers and end users. These small releases are delivered on a schedule using iterations that typically last between one and four weeks each. Plans, requirements, design, code and tests are created initially and updated incrementally as needed to adapt to project changes. Software functionality progress can be checked and monitored much more frequently rather than at end of long milestones.
5. Flexibility of Design: Flexibility defines ability to change directions quickly. As handling change in requirements is the main feature of agile methodology, design has to be made flexible that can handle changes easily. Flexibility is based on the development process used for the project.
6. Improvement in Quality: Test-driven development and refactoring is used. Refactoring leads to higher code reuse and better quality. All aspects of software are improved, from design and architecture to performance of the products of each sprint. Improved communication leads to faster turnaround time for blocking bugs.

V. LIMITATION OF LIGHTWEIGHT METHODOLOGY

a) Main emphasis is on development rather than design and user. It basically focuses on processes for getting requirements and developing code and does not focus on product design.
b) High testing lead times and low test coverage.
c) Many teams requiring high coordination and communication from project managers.
d) Does not scale well to large projects, as numerous iterations are needed to complete the desired functionality.

e) Too much time may be devoted to any single, small feature.
f) On a large scale project, opportunity cost to employ agile methods may be too high for a foregone production on more profitable and lean projects.
g) Management Overhead is increased because a successful application of an agile methodology relies heavily on strong teamwork, the project manager must remain involved in the dynamics of the team.

VI. CONCLUSION

Lightweight software development stresses in - evolving requirements accomplished by direct user involvement in the development process, rapid iterations, small and frequent releases. The improvements in software development process include more stable requirements, earlier fault detection, less lead times for testing, increased communication, and increased adaptive capacity. Different methodologies require different changes to the management and software development cultures .There are number of factors that can directly and indirectly influence the development projects in agile framework. Adopting agile development methodologies has a positive impact on both the productivity and the quality. Hence, development team and customer both are satisfied with its implementation in software development processes.

REFERENCES

[1] Andrew Begel, Nachiappan Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study", First International symposium on empirical software engineering and measurement, pp. 255-264, 2007.
[2] Peter Maher, "Weaving Agile Software Development Techniques into a Traditional Computer Science Curriculum", Proc. of 6th IEEE International Conference on Information Technology: New Generation, pp. 1687-1688, 2009.
[3] Anfan Zuo, Jing Yang, Xiaowen Chen, "Research of Agile Software Development Based on Formal Methods", International Conference on Multimedia Information Networking and Security, pp. 762-766, 2010.
[4] Michael J Rees, "A Feasible User Story Tool for Agile Software Development", Proc. Of 9th Asia-Pacific Software Engineering Conference (APSEC' 02), 2002.
[5] Outi Salo, Pekka Abrahamsson, "Integrating Agile Software Development and Software Process Improvement: a Longitudinal Case Study", pp. 193-202, 2005.
[6] Richard Mordinyi, Eva Kuhn, Alexander Schatten, "Towards an Architectural Framework for Agile Software Development", 17th IEEE International Conference and workshops on Engineering of Computer Based Systems, pp. 276- 280, 2010.
[7] Ying Wang, Dayong Sang, Wujie Xie, "Analysis on Agile Software Development Methods from the View of Informationalization Supply Chain Management", 3rd International Symposium on Intelligent Information Technology Application Workshops", pp. 219-222, 2009.
[8] Pirjo Nakki, Kaisa Koskela, Minna Pikkarainen, "Practical model for user-driven innovation in agile software development", Proc. Of 17th International Conference on Concurrent Enterprising, pp. 1-8, 2011.
[9] Agile Alliance, http://www.agilealliance.org/
[10] Agile Manifesto and Agile Principles, http://agilemanifesto.org/
[11] "Agile Software Development" Wikipedia, http://en.wikipedia.org/wiki/Agile_software_devel opment
[12] Behrouz Far, "Software Reliability Engineering for Agile Software Development", pp. 694-697, IEEE 2007.
[13] Markus Kohlbacher, Ernst Stelzmann, Sabine Maierhofer, "Do Agile Software Development Practices Increase Customer Satisfaction in Systems Engineering Projects?" IEEE International Systems Conference (SysCon), pp. 168 - 172 IEEE 2011.
[14] Sharifah Syed-Abdullah & Mike Holcombe & Marian Gheorge, "The Impact of an Agile Methodology on the Well Being of Development Teams", Empir Software Eng, pp. 143-167, Springer 2006.
[15] Linda Rising and Norman S.Janoff, AG Communication Systems, "The Scrum Software Development Process for Small Teams, IEEE Software July/August 2000.
[16] Nuno Jardim Nunes, João Falcão e Cunha" Wisdom - Whitewater Interactive System Development with Object Mode Version 4.0 / 21 April 2000.
[17] Rietmann: DSDM in a bird's eye view, DSDM Consortium, p. 3-8 (2001).
[18] Cook, J.E., and A.Wolf, "Discovering models of software processes from event-based data", ACM Trans. Softw. Eng. Methodol, 7(3), 215 – 249, (1998).
[19] "Scrum Agile Model" available at: http://www.rightwaysolution.com/scrum-agile development- model.html accessed on: 24/04/2011.