International Journal of Advances in Software Engineering & Research Methodology– IJSERM
Volume 1 : Issue 2

Publication Date : 25 June 2014

Agile Requirements Engineering

[Murdaca, C]

***Abstract*—The Agile Software Development Methodology is fast becoming the more popular and eagerly embraced software development methodology. It is underpinned by the Agile Manifesto which clearly articulates the methodology's underlying principles and values. Requirements engineering is a key component of any development life cycle, yet it has gained little attention in the studies conducted in the Agile Movement. This paper aims to introduce the key features of requirements engineering in an agile setting and how requirements engineering is carried out in two of the more popular agile methods Scrum and Extreme Programming.**

Keywords—agile, analaysis, development methodology, requirements engineering

I. Introduction

The two main software development methodologies employed across industry at present are Agile and Waterfall. The Waterfall methodology is seen as the traditional, robust and sequential development life cycle. Its great strength is that it is supremely logical – think before you build, write it all down, follow a plan, and keep everything as organized as possible [1]. However its sequential nature means that it is resistant to change. Changes to requirements and scope, are hard to support and manage within a waterfall methodology.

Agile systems development methods emerged as a response to the inability of previous plan-driven approaches to handle rapidly changing environments [2]. The Agile methodology is seen as the new, fun, interactive as well as iterative methodology that healthily manages change, and due to its small iterative nature of development, provides tangible deliverables more quickly than otherwise could be achieved in a waterfall approach.

When it is appropriate, an iterative, agile methodology is preferred because business stakeholders are more likely to get what they really want when the solution comes through a process of trial-and-error rather than from abstract preconception of what the requirements might be [3]. Due to the ever changing and fast paced nature of development environment's today, the need to get to market quickly and efficiently is making the agile methodology more popular and therefore more commonly the methodology of choice.

There are a number of agile methodologies that are employed across the industry at present. It is an exciting and ever changing landscape that continues to evolve. A great deal of focus has been placed on new concepts that agile methods have introduced to the project life cycle, however little attention has been given to requirements engineering.

Murdaca, C
MATHS I C
GPO BOX 1771 SYDNEY NSW 2001, AUSTRALIA

The objective of this text is to review the role of Requirements Engineering in two of the most popular Agile Methods, Scrum and Extreme Programming.

The remainder of this text will be set out as follows. Section 2 will review The Agile Manifesto. Section 3 will introduce and define key agile concepts. Section 4 introduces the concept of an Agile Method and some of the more commonly employed methods in use today. Section 5 reviews the benefits of employing agile methods. Section 6 looks at the role of requirements engineering in a software development lifecycle and the role it plays in two of the more popular agile methods; Scrum and Extreme Programming. Section 7 draws the conclusion.

II. The Agile Manifesto

In 2001, seventeen keen agile enthusiasts came together to brainstorm and ultimately develop a Manifesto for Agile Software Development, or as it is commonly referred to as the Agile Manifesto.

The Manifesto has become an important foundation of the Agile Movement, in that it characterizes the values of Agile Methods and how Agile distinguishes itself from traditional methods [4]

At the core of the Agile Manifesto are the following four underlying values [5]:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The Agile Manifesto is also underpinned by the following twelve underlying principles [5]:

5. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
6. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
7. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
8. Business people and developers must work together daily throughout the project.
9. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
10. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
11. Working software is the primary measure of progress.
12. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

13. Continuous attention to technical excellence and good design enhances agility.
14. Simplicity—the art of maximizing the amount of work not done—is essential.
15. The best architectures, requirements, and designs emerge from self-organizing teams.
16. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

III. Key Concepts

Agile methodologies introduce some interesting new concepts that are not used within the traditional waterfall methodology. Some of the key concepts that are at the heart of agile methods are defined as follows [6]:

Agile - refers to software development methodologies that value and support evolving requirements through iterative development, direct Customer/Developer communication and collaboration, self-organizing cross-functional teams, and continuous improvement through frequent inspection and adaption.

Burn Down Chart - a simple, easy to understand graphical representation of work remaining versus time remaining. They are effective for communicating progress and predicting when work will be completed.

Daily Stand-Up - a brief, daily communication and planning forum, in which Agile teams come together to evaluate the health and progress of the iteration. It is a tightly focused and time boxed meeting where team members will focus on answering the following three questions only:

1. What did I accomplish yesterday?
2. What will I commit to, or complete, today?
3. What impediments or obstacles are preventing me from meeting my commitments?

Iteration / Sprint - a predefined, time-boxed and recurring period of time (generally less than 6 weeks duration) in which working software is created.

Product Backlog - a prioritized and estimated list of all outstanding product/project requirements, features, defects and other work items.

Story Points - unit-less measures of relative size assigned to requirements for functionality. Allows the team to focus on the pure size and complexity of delivering a specific piece of functionality rather than trying to perfectly estimate duration of time required for the completion of the functionality.

Task Boards –visual communication and planning tools that are extremely useful for teams working in co-located environments. Due to their intuitive and simple nature, tasks boards provide a powerful means of measuring and communicating iteration health and progress.

User Roles - describe the unique perspectives of the different consumers that will interact with the working software.

User Stories - simple, brief and concise statements, used to describe customer software requirements, from a particular users' perspective.

Now that the key agile concepts have been introduced, the next section will look at agile methods and some of the more commonly employed agile methods in use in the industry at present.

IV. Agile Methods

Agile methods encourage more-collaborative development than do traditional approaches [7]. Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaption, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software and a business approach that aligns development with customer needs and company goals [8]

There are numerous methods employed across the industry today that are 'agile' in nature. Some of the most commonly employed agile methods are listed below:

- Adaptive Software Development (ASD)
- Agile Unified Process
- Crystal (sometimes referred to as Crystal Clear)
- Dynamic Systems Development Method (DSDM)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Kanban
- Lean Software Development
- Pragmatic Programming
- Scrum

The next section will review some of the key benefits of agile methods in general.

V. Benefits

Various studies on agile methods have been conducted across the industry and documented in published literature such as [9], [10] and [11].

The key findings of these studies have been summarized, and are detailed as follows:

- Business and technical risks addressed explicitly during the initial inception and elaboration phases
- Changes can be incorporated more easily and demonstrate business value more efficiently
- Customer satisfaction with opportunities to get and give feedback
- Defects can be removed earlier
- Development practices are easy to adopt and work well
- Greater customer collaboration, leading to better understanding of customer expectations
- Greater predictability of the project execution due to known business and technical risks
- Greater support and opportunity to undertake beta testing
- Increased interaction between the customer and the development team
- Iterations provide smaller control units to program managers for greater visibility and control
- More frequent testing is carried out

- Multidisciplinary work of the team leads to better problem understanding
- Workable software can be delivered earlier and on a more frequent basis

As shown there are numerous documented benefits of agile methods, which further reinforce the growing popularity of the methodology in the industry at present. However, one of the key components to any development methodology is the ability to elicit, document and manage requirements through some formal or informal analysis activity. This formal or informal analysis activity is more commonly referred to as Requirements Engineering.

In the next section, we will review the changing nature of requirements engineering, from a more traditional plan driven approach in waterfall methodologies, to an iterative change driven approach in agile methodologies.

VI. Requirements Engineering

Within a Waterfall approach to development, the majority of requirements engineering will occur within the initial Requirements Phase. All formal requirements engineering activities will be conducted within this phase by the business analyst(s). They will undertake requirements engineering tasks such as:

- Eliciting
- Documenting
- Reviewing
- Obtaining Sign Off
- Maintaining
- Managing
- Tracing Through to End Deliverable(s)

Note the waterfall methodology is documentation heavy, and hence many requirements engineering artifacts are expected as deliverables from the business analyst. Some of the artifact deliverables that will capture the business requirements include, but are not limited to:

- Stakeholder Requirement Documents
- Business Requirement Documents
- Solution Requirement Documents
- Functional Specification Documents
- Process Maps
- Traceability Matrices

Although requirements engineering is isolated to the initial phase in a waterfall methodology, it does not preclude that requirements do not arise post this phase. Although every endeavor will be done to ensure that all requirements are captured in this phase, sometimes requirements will be missed, or only identified later, in subsequent phases. When requirements are identified in subsequent phases, the waterfall methodology employs a change management process to capture these requirements. The change management process is utilized to formally investigate the scope, priority and *real* business need for the requirement and the implications of the requirement being met within the scope of the current or subsequent project deliverables.

Requirements engineering within an agile methodology is strikingly different. Agile by its very nature is not documentation heavy and development is performed through short, time-boxed iterations. Requirements engineering is still a very important component. If requirements are not identified and prioritized how can project deliverables be articulated and therefore met? Requirements engineering in Agile methods is not as formal, nor documentation heavy as in Waterfall methods, however it is still a very important part of the development life cycle.

In agile development methods, requirements engineering processes are not centralized in a single iteration, nor is it restricted to being conducted prior to development commencing. It is more iterative and evolves throughout the project lifecycle. After a streamlined planning, requirements definition and solution phase is completed to get the project underway, iterations of more detailed planning, requirements, design, build and test take place in waves. This approach allows for immediate modifications of the product as requirements come into view [13].

As introduced in Section IV, there numerous agile methods that are utilized in the industry at present. Two of the more popular agile methods employed across the industry at present are Scrum and Extreme Programming. Due to the varying and informal nature of how requirements engineering is conducted in an agile method, it is worth reviewing how requirements engineering is conducted by a business analyst in two of the more popular agile methods, Scrum and Extreme Programming.

The next two subsections will now introduce the key features key features and review how requirements engineering is conducted in the two agile methods Scrum and Extreme Programming.

A. *Scrum*

Scrum is one of the more popular agile methods. Scrum structures development in cycles of work called Sprints (also referred to as iterations). These sprints generally have duration of between one to four weeks. The sprints are time boxed, that is, they end on a specific date whether the work has been completed or not, and are never extended [1]. At the end of each sprint, a working increment of the software is delivered [6]. Requirements engineering is performed and captured within what is referred to as a *product backlog*. The product backlog is a list of functional and non-functional requirements, prioritized in order of importance to the business, together with any issues, dependencies and estimates.

The product backlog can be articulated in any way that is clear and sustainable, though either Use Cases or User Stories are often used to describe the product backlog items in terms of their value to the end user of the product [14]. As highlighted in [14], it is good practice and a valuable exercise if five to ten percent of each Sprint is dedicated to refining the product backlog, to ensure it is up-to-date, as the requirements may need to change as the product deliverables continue to evolve as each sprint is completed.

It is important, within the Scrum agile method, that requirement engineering is performed at the start of the project, and then continually during each sprint undertaken. This will ensure that the requirements contained within the product backlog are detailed, up-to-date, split into smaller requirements were possible, and that associated estimations are current and valid.

B. *Extreme Programming*

Extreme Programming (XP) is one of the more popular agile methods in the industry at present. XP seeks to improve software quality by focusing on technical excellence, while improving project agility and responsiveness to changing requirements by valuing small yet frequent, time-boxed releases.

XP is underpinned by the following five key principles [6]:

1. Communication
2. Simplicity
3. Feedback
4. Respect
5. Courage

It is also characterized by the following six phases [15]:

1. Exploration
2. Planning
3. Iterations to First Release
4. Productionising
5. Maintenance and Death

Extreme Programming advocates requirements engineering throughout the development life cycle in small, informal stages [12]. Similar to other agile methods, documentation in XP is kept to a minimum and as such the main form of documenting requirements in XP is undertaken via user stories and acceptance test cases [16].

The next section will summarize this text and draw the conclusion.

VII. Conclusion

Requirements engineering is the critical initial phase within the waterfall development methodology. It employs a very structured and formal approach to eliciting, documenting and managing requirements. As agile methods prove ever more popular it is clear that requirements engineering still plays a critical part in the development life cycle. In agile methods, requirements engineering is not centralized in a single iteration, nor is it restricted to being conducted prior to development commencing. It is more iterative and evolves throughout the project lifecycle.

There are various agile methods currently employed in industry at present, they all have their pros and cons, however they are all founded on the underlying principles and values of the Agile Manifesto. Both these methodologies are focused on short, incremental, iterations that offer small incremental product tangible deliverables at the end of each iteration. Neither method employs a formal requirements engineering process. However requirements engineering performs a valuable role in both methods.

Requirements engineering in Scrum and XP is performed both at the start of the project, and iteratively throughout the sprints / iterations undertaken. Neither method employs a formal framework for documenting these elicited requirements. However given the emphasis on smaller, faster and more visible deliverables to customers in an agile framework, it is not surprising that the preferred technique for documenting and communicating requirements in the two methods is very similar, as they are captured through Acceptance Test Cases, Use Cases and User Stories. This allows requirements to more easily adapt and evolve during project iterations. This is in sharp contrast to the more formal requirements analysis phase conducted within a waterfall methodology, whereby changes to requirements post this phase is managed via a change management process.

As shown in this text, it is clear that although the approach employed for requirements engineering differs between Waterfall and Agile Methodologies, it is clearly similar within different Agile Methods.

References

[1] Deemer, P, Benefield, G, Larman, C, Vodde, B *The Scrum Primer* 2010
[2] Highsmith, J *Agile Software Development Ecosystems*, Addison-Wesley, Boston, USA 2002
[3] Podeswa, H, *A Practical Guide to Requirements Gathering Using the Unified Modeling Language*, Course Technology CENGAGE Learning, Boston USA 2010
[4] Cohen, D, Lindvall, M, Costa, P *An Introduction to Agile Methods*, Advances in Computers, Vol 62, pp 1 – 66 2004
[5] *The Agile Manifesto* www.agilemanifesto.org
[6] Davis, S, *The Agile Glossary of Terms*, WhitePaper, 2010 Davisbase
[7] West, D, Grant, T *Agile Development: Mainstream Adoption Has Changed Agility*, Jan 2010
[8] Kaur, R, Choudhary M, Mehta, R, *Agile Process: An Enhancement to The Process Of Software Development* IJCSNS International Journal of Computer Science and Network Security, Vol 12, No 7 July pp 101 – 106 2012
[9] Abrahamsson, P, *New Directions on Agile Methods: A Comparative Analysis*, Proc. 25th Int'l Conf. Software Eng. (ICSE 03), IEEE CS Press, 2003, pp. 244–254.
[10] Cohen, D, Lindvall, M, Costa, P *An Introduction to Agile Methods*, Advances in Computers, Vol. 62: Advances in Software Engineering, M.V. Zelkowitz, ed., Elsevier, 2004, pp. 1–66.
[11] Dybå, T and Dingsøyr, T *Empirical Studies of Agile Software Development: A Systematic Review*, Information and Software Technology, vol. 50, nos. 9–10, 2008, pp. 833–859
[12] Cao, L, Bamesh B, *Agile Requirements Engineering Practices: An Empirical Study*, IEEE Software 2008 pp 60 – 68
[13] Hass, K, B, *The Blending of Traditional and Agile Project Management*, PM World Today May 2007 Vol IX Issue X
[14] Sutherland, J, Schwaber, K, *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework* 2011
[15] Beck K. 2000. *Extreme Programming Explained*, Addison-Wesley Pearson Education, Boston.
[16] Duncan, R, *The Quality of Requirements in Extreme Programming, Software Development Methodologies*, June 2001