© 2017 IJSRST | Volume 3 | Issue 1 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

Is Agile Project Management fit for small tech start-ups?

Priti Asthana
Project Manager, Pittsburgh, PA, USA

ABSTRACT

Like other professions, project management is evolving, with approaches to project management emerging. As alternatives to the traditional project management methodology, agile software development life cycle models provide innovative approaches to software development. Agile approaches are incremental in nature, with a primary focus on the deliverables of the iterations. These approaches have proven effective in highly complex projects, characterized by high uncertainties and large enterprises have adopted them, with considerable success. While the use of use of agile approaches to project management has brought in large organizations, their applicability in small startups is questionable. This is largely due to the differences between large and small enterprises. Small scale startups have limited funding, limited number of developers and are constantly faced with the constraints of time and cost and may not be able to use agile approaches with ease. Through a critical examination of the benefits and disadvantages of agile approaches to software development, particularly Scrum and Extreme Programming, this paper explores the possibility of extending agile approaches to small startups. The paper concludes that agile approaches to project managers are not fit for small tech startups.

Keywords: Agile methodology, Scrum, Extreme Programming, Project Management

I. INTRODUCTION

Agile approaches to project management have become popular in the Information Systems (IS) today. Agile methods are an alternative to the old-style software development approaches such as the waterfall model. The wide adoption of agile approaches to software development is attributable to their recognition of the interaction of people and development team as the main factor influencing project success, combined with an intense emphasis on efficiency and maneuverability (Cockburn & Highsmith, 2001).

Central to agile methodology is the use of incremental and iterative development process (Yau & Murphy, 2013). This is illustrated in figure 1. The aim of agile methodology is to plan out and deliver small portions of the project at a time rather than defining and documenting the whole project during the planning phase like the waterfall SDLC. However, agile methodology will follow similar phases to waterfall model but will have loops between the phases. According to Yau and Murphy (2013), agile development process begins with the basic set of deliverables, followed by planning, implementation and testing of other components in the subsequent iterations, as illustrated in figure 2.

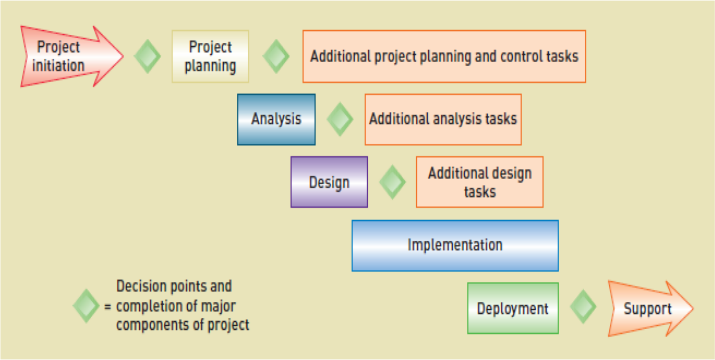

Figure 1: Iterative incremental software development life cycle. Retrieved from Satzinger, Jackson, and Burd (2016).

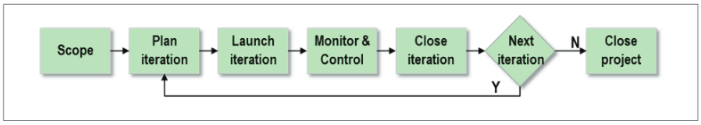

Figure 2: Iterations in agile project management life cycle. Retrieved from Wysocki (2009).

Figure 2 shows the iterations of the project. Each phase in figure 1 is characterized by iterations illustrated in

figure 2. As such, the focus in agile software development is in short deliverables that are tested before proceeding to the next step rather than testing the products at the end of the project. This practice helps to keep bugs and code errors minimal throughout the project life cycle. Agile methodology has proven to be very useful in complex projects surrounded by significantly high levels of uncertainties, and has been widely adopted in big organizations. The current paper discusses agile approaches, mainly scrum and extreme programming, and critically assesses the possibility of effective applicability of agile methodologies in small enterprises.

II. METHODS AND MATERIAL

A. Principles of agile project management approaches

Agile approaches to software development are built on twelve principles. These include:

1. Satisfaction of client through early and incremental delivery of high-quality software. This is achieved through proper contractual relationship between the development team and the client, and incremental iterations, where client can use some deliverables of the project at the end of early iterations (Stare, 2013). As such, successive iterations are considered to add value to the software, which encourages clients to prolong the project. However, this is possible in certain project, especially those involving high level of uncertainties.
2. Harness change even in late development to increase customer's competitiveness. Agile methodologies allow changes to project scope at any stage of the project life cycle. This feature is beneficial especially when the project is complex, making identification and defining all project risks impractical before its initiation.
3. Incremental development of the software. This principle requires that the project team delivers working software at short intervals.
4. Collaboration of the project team and the client throughout the project life cycle. Engagement of customers to the software development is to ensure that the developers clearly understand the requirements of the product as well as clear communication of changes during the software development.
5. Offer support and provide a conducive environment for the developers. This principle relates to the motivational theory.
6. The most effective method of communicating within a development team is face-to-face communication. This principle requires project team members to have strong interpersonal verbal communication skills.
7. Functional software is the main measure of progress. Based on this principle, agile approaches focuses on deliverables of the iteration. This helps the developers to avoid delivering a non-functional product at the end, which is the greatest risk in software development.
8. Agile methodologies promote sustainable development.
9. Continuous focus on technical excellence and excellent software design enhances agility. This means that agility demands developers with high technical know-how to allow agile thinking (Stare, 2013).
10. Simplicity in planning the iterations. This principle is made to ensure efficient achievement of the iteration goals.
11. The best system architectures and designs develop from self-organizing teams. This principle suggests that development team is able to organize their work without the intervention of the project manager. In other words, autonomy of the development team may lead to excellent job.
12. Regular review of the software improves the effectiveness of the team. This principle requires the developers to review the software at intervals so as to adjust its work accordingly.

B. Agile project management approaches

The agile software development manifesto consists of several methodologies to agile project management. Some agile software development approaches include SCRUM, Extreme Programming, DSDM, Crystal, Feature-Driven Development, Adaptive Software Development, and Pragmatic Programming (Stare, 2013). This paper will focus on SCRUM and Extreme Programming approaches.

SCRUM

Since its introduction in 1996, scrum has become one of the widely used agile approaches to software development. Scrum is inspired by empirical inspect and adapt feedback loops to address project complexity and unforeseeable risks. To achieve this, scrum follows iterative and incremental product development framework. In this approach, the software development process is divided into short iterations called sprints.

The development process involves three stages; pre-sprint planning, sprint and post-sprint meeting (Yau & Murphy, 2013). During the pre-sprint planning, the project team identifies and selects features and functionalities from a backlog. Also, planning and prioritization of the collection features is done at this stage. In the sprint stage, the development team selects the features it wishes to work on and start the development process. The project team holds a meeting every morning before starting the work to ensure effective communication between the development team and the product manager. A sprint lasts between one to six weeks (Yau & Murphy, 2013). Importantly, the product is maintained in a shippable state throughout the project lifecycle. In other words, the product is properly integrated and tested throughout the project life cycle. The product is reviewed after every sprint during the post-sprint meetings.

Scrum roles

Scrum has three roles: scrum master, product owner and development team. The scrum master serves as the process owner and process manager. He or she ensures that all project activities are understood and support the team through facilitation and coaching. Therefore, the core responsibility of the scrum master is to remove any obstacle that may hinder the project team from achieving its sprint goals (James, n.d). This makes achieving each sprint deliverables realistic and visible to the product owner.

Product owner is the custodian of the project functionalities. He or she gives the requirements of the product in line with the organization needs (James, n.d). As such, the product owner is engaged with the development team to continuously communicate the needs of the product. Interestingly, it is sometimes difficult to strike the right balance of product owner involvement because this model encourages self-organization of the team while demanding the presence of the product owner so as to respond to arising questions from the team.

The development team is the last role of scrum. Its responsibility is to transform product requirements into deliverables that build potentially releasable product. Usually, the development team is a small (3 to 9 members), self-organized and cross-functional unit which jointly accounts for its work. For software development projects, a typical team will include software engineers, architects, systems analyst, programmers, quality assurance expert and testers (James, n.d).

C. Applicability of Scrum in small enterprises

The roles and processes of Scrum may have several benefits to small enterprises. First, the daily meetings can improve communication between the project manager and the development team members (Yau & Murphy, 2013). Effective communication between the project team members and project master may decrease time and cost due to possible communications. Additionally, effective communication may result in high-quality product since excellent software can be designed only when each member of the development team understands the overall scope of the project and how other members are implementing certain parts (Yau & Murphy, 2013). Given that project scope in small startups keep on changing than in large enterprises (Yau & Murphy, 2013), changes to the software structure should be well communicated to achieve high-quality software.

The pre-sprint planning, on the other hand, helps project team to narrow down their activity list and concentrate on the immediate goals. This feature is essential to small enterprises because the end product is often not completely defined; thus, it is very easy for the development team to get trapped in the development of too many features rather than focusing on the critical features (Yau & Murphy, 2013). By dividing the tasks into sprints, the development team can focus on iteration goals and deliver main features step-wisely.

III. RESULTS AND DISCUSSION

A. Extreme Programming

Extreme Programming is the other widely used agile software development approach. Extreme Programming is based on dynamism of project requirements, short development cycles (Yau & Murphy, 2013), virtual teams, changing technologies and collaborative participation of all stakeholders to project development (Thomsett, 2002). According to Thomsett (2002), the relationship between the product owner and the project team is critical project success using this approach.

Extreme Programming follows a test-driven development process and emphasizes that the programmers write acceptance test for the code before implementing the features, whose benefits in software development cannot be underestimated. Writing the test cases before implementing features helps to determine if the features fulfill the specifications as defined before project initiation (Yau & Murphy, 2013). Again, test-driven development helps to reduce bugs and code errors. Further, writing test cases before implementing the features helps to easily determine if changing parts of the code will affect other sections by simply running the test suite. Overall, Extreme Programming increases the quality of the product and decreases the time and cost associated with debugging at the end of the project.

B. Applicability of Extreme Programming in small enterprises

Despite Extreme Programming promising to yield high-quality product, it has limited applicability in small enterprises for several reasons. First, small scale enterprises have limited number of software developers. Notably, developers in small enterprises and small startups are constantly changed (Yau & Murphy, 2013). As such, it is not worth spending much time in writing comprehensive test cases for all features before implementation. It is possible that the client may change their requirements before writing the test cases is complete, rendering such tests useless.

Second, it is very common for small startups to ask for a few features as prototypes to test some ideas so as to make their final decisions. In this scenario, it is not reasonable to write all the test cases. It is more meaningful to implement such prototypes fast so as to speed up the decision-making process (Yau & Murphy, 2013). As such, the use of Extreme Programming in small startups is limited.

Third, Extreme Programming is limited to small startups due to shortage of funds. Due to shortage of funding, most small scale enterprises will focus on minimal viable product rather than the quality of the product (Yau & Murphy, 2013). All they may need is at least functional product to present to investors. In case an enterprise decided to fund Extreme Programming and a close competitor released a similar product, much of its product may not help much, translating to loss of its money value (Yau & Murphy, 2013).

The other aspect that limits the applicability of Extreme Programming is its focus on pair program, which requires two programmers to write code together on the same computer (Yau & Murphy, 2013). Pair programming is used to writing better code. The idea behind pair programming is that when two developers are combined, they are likely to share knowledge and produce high quality code besides collective ownership of the code. The programmers are more likely to overthink a simple problem; leading to high quality code than if they were alone, which is critical in big companies. Small startups may not reap the benefits of pair programming. First, small enterprises have a limited number of developers. Secondly, small startups are significantly challenged by constraints of time and cost; thus, improving the quality by doubling the cost (hiring of a second developer) could be impossible for small enterprises. Again, the small number of programmers dictates that each developer is assigned to a particular area, making it impossible to enjoy the benefits of pair programing.

IV. CONCLUSION

Agile approaches to project management offer alternatives to the traditional software development approaches such as the waterfall model. Agile approaches divide project activities into iterations, with a primary focus on incremental delivery of the product. Scrum and Extreme Programming are typical examples of agile approaches to project management. While agile

approaches to project management have become very popular in the IS field, their applicability in small startups is limited. Small startups are faced with cost and time constraints, unable to manage the costs associated with agile methodologies. Again, the requirements of small startups often change constantly and may be costly if Extreme project management approach used. If a customer changes a mind before the test cases are over, such test cases become useless. Further, small startups have a limited number of developers, making it impractical to implement pair programing emphasized by Extreme Programing approaches.

V. REFERENCES

[1]. Cockburn, A. and Highsmith, J. (2001). Agile Software Development joins the "would-be" crowd. Cutter IT Journal. 34(9), 122 James, M. Scrum Methodology: An Empirical Framework for Learning (Not a Methodology).

[2]. Satzinger, J., Jackson, R., & Burd, S. (2016). *Systems analysis and design in a changing world*. Boston: Cengage Learning.

[3]. Stare, A. (2013). Agile project management – a future approach to the management of projects? *Dynamic Relationships Management Journal*, *2*(1), 43-53.

[4]. Thomsett, R. (2002). Radical Project Management. Upper Saddle River (NJ): Prentice Hall PTR

[5]. Wysocki, R. (2009). *Effective project management: traditional, agile, extreme* (5th ed.). Indianapolis: Wiley Publishing.

[6]. Yau, A., and Murphy, C. (2013). Is Rigorous Agile Methodology the Best Development Strategy for Small Scale Tech Startups?