Incorporating Agile Methods into the Development of Large-Scale Systems

Trenton Hafterson
University of Minnesota, Morris
600 East 4th Street
Morris, MN 56267
haft0004@morris.umn.edu

ABSTRACT

Many small-scale developers have shifted from a traditional, waterfall method for developing software to lighter weight, agile methods. However, this shift has been difficult for large-scale projects. In this paper, we introduce some approaches for scaling agile methodologies to large-scale development projects. First, we define the basic framework of both waterfall and agile methodologies with how they relate to large-scale systems. Then we explain the advantages and disadvantage of large-scale systems using extreme programming. We present some solutions to work around such disadvantages seen in XP, such as hybrid methods involving both agile and traditional approaches or using a framework that supports a slower introduction into an agile methodology. Lastly, we give an overview of tailoring of methods to individual systems since each system is unique and there is no right solution to every situation.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Life cycle, Software configuration management, and Software process models*; D.2.1 [**Requirements/Specifications**]: Methodologies and Tools

General Terms

Management, Design, Documentation

Keywords

Agile, Extreme Programming, Waterfall, Soft-Structured Agile Framework, Agile Software Solution Framework, Software Design

1. INTRODUCTION

Software development is a complex process that can be accomplished in many different ways with varying results. Ideally, to be a successful software developer one must be able to build a quality project in a reasonable time frame and budget for the client company. There have been many different approaches engineered for various situations. Some methodologies that have been becoming more common in practice are those known as agile software development. These methods are showing convincing benefits to small to medium scale companies and should not be ignored by global companies developing large-scale systems [5]. This paper presents and analyzes issues and potential paths to take advantage of scaling agile development processes to work in large-scale systems.

2. BACKGROUND

Today there are many approaches to software development, but in practice few are seen as applicable to large-scale systems. As most agile methods and frameworks are commonly used in small-scale to medium-scale systems, it is unclear what can work for large-scale systems [10]. Before we describe some of the challenges of using an agile approach in large-scale systems, we will describe and contrast a traditional waterfall method and the agile development philosophy.

2.1 Waterfall Method

The waterfall method is a sequential design process in which each stage is completed before proceeding to the next one. An implementation of the process includes five phases: requirements specification, design, implementation or coding, testing and debugging, and maintenance [11]. This process originates from the manufacturing and construction industries. An example of a similar phased process would be constructing a building: customer requirements, blueprints, construction, moving in, and maintenance. In is not possible to move into a building before the construction phase. But, this rigid ordering is not necessary for software development. Software can be more flexible because it can be broken down into smaller subsections that may not be dependent on other sections.

2.2 Agile Development

The more programming methods evolve to suit the environments of software development, the less they resemble the traditional waterfall methods [11]. Agile development is a way of thinking about development. It is not a method in itself, but rather a philosophy [8]. This philosophy is focused on a set of 4 basic values and 12 principles, as stated in the Agile Manifesto [2].

UMM CSci Senior Seminar Conference Morris, MN.

2.2.1 Agile Manifesto

The Agile Manifesto is a document written by 17 software developers in the quest to find a lightweight, effective development method. The Agile Manifesto's writers include representatives from many of what are now known as agile methods or practices. Using a collective knowledge of software development and seeing a need to change from heavyweight methods such as waterfall, they wrote the Agile Manifesto.

The Agile Manifesto values read as follows [2]:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> 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
>
> That is, while there is value in the items on the right, we value the items on the left more.
>
> **Principles behind the Agile Manifesto:** We follow these principles:
>
> - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
> - Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
> - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.
> - Business people and developers must work together daily throughout the project.
> - Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
> - The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
> - Working software is the primary measure of progress.
> - Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
> - Continuous attention to technical excellence and good design enhances agility.
> - Simplicity–the art of maximizing the amount of work not done–is essential.
> - The best architectures, requirements, and designs emerge from self-organizing teams.
> - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

This philosophy influenced the creation of many different agile methods seen today. The most notable one that we will describe is *Extreme Programming* (XP).

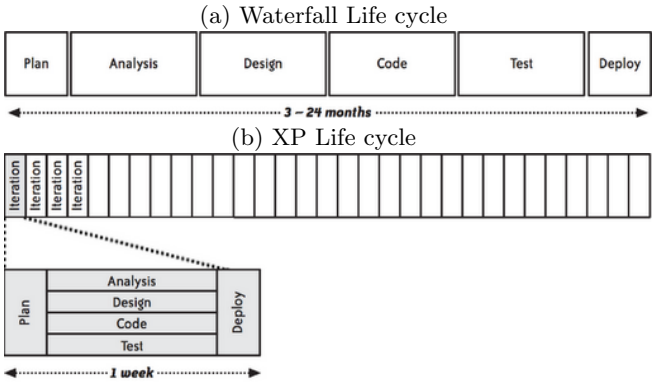

Figure 1: This figure compares the waterfall and XP life cycles [8].

2.2.2 Extreme Programming

Extreme programming is one of the most popular of today's agile methods. Focusing its values on communication, simplicity and feedback to improve the speed of development and quality of code, it eliminates the requirements, design, and testing phases, and all the extensive documentation as separate phases, but not entirely [5]. Rather, XP suggests integrating all of these steps at the same time in short iterations of about one to two weeks (see Figure 1). To do this, XP suggests that developers keep constant communication with the client company or customer, thus allowing flexibility that is impossible in rigid waterfall-like processes. The customer is considered a part of the team, and works with the developers creating user stories[1], developing tests, and prioritizing features to point the project in the right direction [8]. Each iteration consists of a little of each of what is normally seen in traditional phase-based waterfall methods (see Figure 1). Each iteration includes a little planning, analysis, design, coding, and testing, followed by deployment. Each iteration deployment is a point that the customer may choose to *release* the project for use by the client company. Another important part of XP is the numerous practices that bring the agile values together. Some of the most notable of these practices include pair programming, and frequent testing and refactoring. For example pair programming is where two programmers work together on one workstation collectively writing and reviewing each other's code and guiding each other. Studies have shown this to result in fast production of simple clean code, that requires less refactoring later [3].

[1]A user story is one or more sentences in the everyday or business language of the end user that captures what the user wants to achieve.

2.3 Benefits of Being Agile

Heavily structured plan-driven methods such as waterfall:

- do not adapt easily to changing requirements,
- rely heavily on the quality of initial plans and estimations, which are often unreliable, and
- lack continuous customer involvement, which can lead to misunderstandings and wasted time.

In the design of software systems, features and functions that seem great to the developers may not always be needed or understood by the customer, and projects may need changes at later stages, when the costs of these changes are the highest. Agile software development aims to remedy the deficiencies of heavyweight methods. With short iterations and regular customer involvement, project changes can be handled at any stage [10]. Also, coding standards, pair programming, and extensive testing seen in most agile methods allow for development of potentially cleaner code, and cleaner code requires less refactoring and documentation.

3. CHALLENGES OF LARGE-SCALE XP

In spite of the benefits of being agile, there are challenges in scaling XP to large-scale systems. In this section, we look at the challenges that exist for global companies developing large-scale systems while using XP. As projects grow to the large-scale development size, the number of challenges in using XP will grow [5]. One common characteristic of large-scale software development is that the project is often distributed over multiple teams, buildings, or countries. In distributed projects, the need for effective coordination grows with the project's complexity. The main difficulties encountered in a distributed development project can be categorized as *spatial*, *temporal*, and *cultural*.

3.1 Spatial Challenges

A critical part of XP is open communication within the development team and with the customer. Both direct communication and the ability to coordinate are limited in the case of a spatial separation. Members of a team that all work in one office see, and work with, each other every day. In the case that a team is separated, these everyday interactions during the XP process are potentially lost. Open communication builds trust, which can help the coordination and effectiveness of the agile process. This trust can be more difficult to establish in a distributed software development scenario [5].

To make the most of agility and XP over a distance, using forms of electronic communication is a requirement. This can range from instant messaging or e-mail to audio and video conferencing systems. As all types of electronic communication have their benefits and drawbacks, a team should not just rely on one. For example, text-based conversation may lack the expression of emotion from voice, facial expressions, and hand gestures, while video and audio may lack the speed and convenience of text-based conversations over a long distance due to transmission delays.

3.2 Temporal Challenges

In a situation where development teams are located in different time zones, communication efforts can suffer. In the scenario where the working hours do not overlap, communication is often completely asynchronous, and even delayed over many hours. This time difference also exacerbates the communication challenges between the development team and the customer. As the customer is a critical part of XP process, it is important that the customer have input at each step of production and assure the project is going in the right direction. The customer is also valuable to the client company, meaning they may not always be available in every XP scenario. In the case of a distributed software development project, it is unreasonable for a customer to be present at all development site locations. In this case, development teams may need to make even further efforts to help continue open communication amongst themselves and with the customer. To achieve a high level of communication in an asynchronous scenario, both the development team and client company need to be highly committed. One approach may be to give a team member the role of moderator to help facilitate communication between sites [5]. This moderator role can also be applied to the customer; a client company can elect a *surrogate* to represent them to make distant interactions more feasible. This surrogate would have extensive knowledge of the client company's interests.

3.3 Cultural Challenges

Lastly, there is the possibility of cultural differences between the teams. The cultural differences may not be directly related to distributed scenarios, but the lack of in-person face-to-face conversations and meetings can add to misunderstandings. These differences can stem from language barriers, cultural references, and customs. It is important to note that even when both teams can communicate in the same language, they may not always speak the same dialect. When face-to-face, these differences can be recognized with the use of verbal and physical gestures in the context of the situation. In a distributed scenario, the cultural differences can become even more challenging. Audio and video conferencing, when available, have the benefit of including emotion, but may not always prevent misunderstandings. This scenario may also have great benefit from the role of a moderator to help bridge the differences [5].

4. HYBRID AGILE METHODS

Although XP was formed for small to medium scale software development projects, it can still be scaled with effort from both client and developer. Most of the principles from XP can be applied to distributed scenarios. With the high level of effort needed from both sides, this may not always be the most reasonable approach [5]. As some specific aspects to the XP method may be more feasible than others, it may be okay to adopt some aspects of XP and find alternative approaches for others.

In practice, it is a great challenge for many organizations to take on a complete agile development approach, and an even greater challenge for a large-scale project [9]. To be successful in a large-scale agile environment one must consider taking alternative approaches to some XP practices and other popular agile development methods.

One can consider adopting a more hybrid method that attempts to use the best of both waterfall and XP-like prac-

tices. As specific agile practices have been shown to be successful for large-scale systems, but not all are potentially suitable, finding a middle ground can be desirable. Practices such as iterative life cycle, pair-programming, and customer involvement can be suitable to large-scale systems. Large-scale systems may find agile practices such as minimal documentation and code refactoring less feasible [9]. A large-scale project may need comprehensive documentation for reasons such as: lack of communication among developers, personnel turnover, and the use of third-party maintenance organizations. Also the fact that the system itself is quite large means that having a somewhat stable core that is not going to be refactored or changed allows multiple teams to make changes that relate to the core simultaneously.

Also large-scale projects tend to have a larger code base and lengthy development cycles, making code refactoring time consuming and costly [9].

4.1 Soft-Structured Agile Framework

One hybrid method is Soundararajan's *Soft-Structured Agile Framework*. This framework consists of two main parts: the *Agile Requirement Generation Model* (Agile RGM) and the Development Process. This particular framework's main objective is to accommodate change in both large-scale or small-scale projects [9]. We will describe the two parts and then summarize the benefits to the soft-structured agile framework.

4.1.1 Agile RGM

The Agile RGM is a set of well-defined activities that provide a more structured approach compared to XP. As shown in Figure 2, the Agile RGM consists of three phases to help capture requirements: Education, Feature Development and Story Development. All of theses phases incorporate agile principles and practices such as customer involvement, iterative life cycle, and minimal documentation. The Agile RGM is employed at the beginning of a project that will later follow either an agile or a conventional development process.

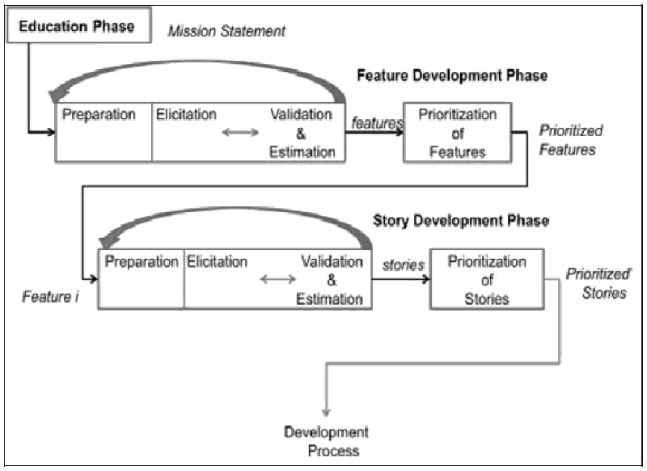

Figure 2: Agile RGM [9].

Education phase.

The education phase is essentially a meeting among the development company, the client company and potentially other various customers to help create a better understanding of the project. This is necessary to help build and plan a set of objectives and goals that are to be achieved in later phases and in the finished product. This is different from the usual XP approach, which employs smaller sessions at the beginning of each iteration.

Feature Development phase.

At this stage, the customer works with the development team to iteratively identify expected system *features*. A feature is a small set of functionality that is valuable to the customer. In creating a feature, the development team will give an estimation of how long it will take or if it is possible. Then after a feature is accepted by both customer and developers, the customer will prioritizes each feature according to its *business value* to the client company. "Business value is something that delivers profit to the organization paying for the software in the form of an Increase in Revenue, an Avoidance of Cost, or an Improvement in Service" [6]. For example, consider development of a website for an e-commerce company. The "Online Payment" feature has a high business value.

Story Development phase.

Using the features created in the previous phase, developers require additional details before proceeding. In this phase, features are decomposed into *stories*. A story is a refined user- or customer- expected feature that will be used in the development process. If the development team is made of multiple teams, each team may independently work towards decomposing one or more features into stories. Here, the feature "Online Payment" is decomposed to "As a user, I can pay by credit card".

4.1.2 Development Process

In this part of the soft-structured agile framework, the developers may take two alternative approaches depending on the scale of the system (see Figure 3). For small-scale systems, the development team may follow an iterative structure like XP, and make each story an iteration. For large-scale systems, the development team may require a more structured approach and can choose to follow a more conventional waterfall-like approach.

For large-scale systems, a waterfall-like approach is usually deployed. First, subsets of stories are transformed into one or more requirements. For example the story "As a user, I can pay by credit card" will be given much more detail and transformed to "the system shall use Advanced Encryption Standard(AES) to encode all credit card information to be transmitted over the internet." The requirements produced from this stage will each be developed in a waterfall-like approach similar to the example seen in Figure 1(a). Although a conventional approach is used here, it still fits within an agile environment as it is guided by the features and stories produced from the Agile RGM process. With the requirements, this stage will also still be able to adapt to changes more easily than conventional waterfall methodologies because requirements are created from stories [9].

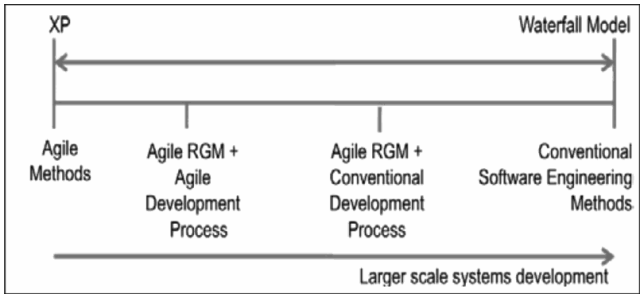

Figure 3: Spectrum of Software Development Approaches [9].

4.2 Benefits of Soft-Structured Agile Framework

The benefits of this framework are that it reflects fundamental agile practices and still meets the needs for large-scale systems. As seen in Figure 3, with the use of the Agile RGM, this approach fits between agile methods and conventional methods. With this approach, the difficulties seen in the XP method can be reasonably avoided, but at the same time agile development practices can be applied. The framework has been presented at Capital One, Richmond, Virginia and was well received. The approach reflects many of the principles and values embraced by that organization. Further on-site testing and research is still required for a formal validation of this approach [9].

5. PHASING INTO LARGE-SCALE AGILE PROCESSES

Another consideration that should not be overlooked is how to implement a method. Most organizations may find it inappropriate to make a full transition to an agile approach in a short period of time. They can consider a method that allows them to phase into an agile approach. For a large-scale system this may be more feasible, not only for the developers, but also for the business value. To adopt new development processes can be challenging and require time. It also can be more attractive from a business viewpoint, because even as being agile has its benefits, a company may not want to risk time and money on something new and unknown. An organization can phase into an agile method by adopting agile processes in a pattern best fit for their existing resources. This can allow for change and fine-tuning of each process within an agile method to best work for that organization. Adopting all aspects of an agile method at once may be too much change, psychologically or technically, for organization to handle without the guidance of an outside expert or mentor who has mastered agile methodologies [7, 8].

5.1 Agile Software Solution Framework

The Agile Software Solution Framework is a set of tools to help a development team during the transition to adopting an agile method. A transition using such framework can take up to a period of 3-5 years [7]. The process can be broken down into three phases.

The first phase of adoption is focused on applying specific agile practices. During this phase a development team will adopt a subset of practices used in an agile method. An example would be that a development team chooses to adopt pair programming, peer review, and code refactoring. As they begin to use these practices, they will then refine them as they see fit to best work for their situation. During this phase, the developers will incrementally introduce new practices after the previous ones are fine-tuned to the developers. The final goal of this phase is to work towards achieving agile development attributes such as: speed, flexibility, and responsiveness.

The second phase is to introduce the key components that differentiate an agile method from traditional methods. In this phase, the development team practices open communication, not just within the team but also with the customer. During this time, the organization values and principles should resemble the Agile Manifesto. An example would be that the team now introduce the customer as a part of each iteration.

The last phase is to continue to fine-tune the agile method that the organization has adopted. During this phase, the development team should work towards mastering the agile method and keeping the agile process. For example at this stage a team works towards quality production with minimal resources within a desired time frame.

5.2 Benefits of Agile Software Solution Framework

The benefit of phasing towards a goal of using an agile development method is that as it is introduced in parts, each aspect has time to be fine-tuned to the situation of the development team. This may later remove the need to tailor a method to fit one's situation. If the 3-5 years suggested is not too long of duration to take to transition, this method has great potential. Two case-studies, both in early phases, each reporting positive feedback and progress with the approach, demonstrate that Agile Software Solution Framework is plausible [7]. Further research and completion of the current case-studies is required for a formal validation of this approach.

6. AGILE METHOD TAILORING

Almost all software development projects are unique, and the choosing of a method, or variant of a method is dependent to the situation of the project [4]. As a result it is uncommon that a method would work exactly as it is originally published.

It is common to tailor a method to a company's specific requirements and resources. Creating a new method of your own is not advised for those inexperienced with agile methods. In such a case, starting with an existing agile method and iteratively tailoring it is recommended [8]. As there is no one way to go about tailoring, practitioners have stated that most of their tailoring efforts were based on intuition and personal experience of the managers or developers involved [4]. With the lack of documentation on how to tailor a method, it is suggested that "unless you understand the rationale, you cannot make an informed decision about extending that step, tailoring it, or dropping it" [4].

For example, an organization should try pair programming before they decide to change it or drop it altogether. A case study of a development team using pair programming tried this practice as defined, but with the wide range of experience levels of their programmers they altered the practice into *enhanced pair programming*. Enhanced pair

programming consists of one senior and two junior programmers. The senior member works between the two junior member's workstations as a navigator, guiding and reviewing for both workstations rather than just one like seen in normal pair programming [7]. Whether one chooses to adopt a fully agile method like XP or a hybrid, and if they plan to phase into such method or implement it all at once, they should consider tailoring it to their situation.

7. CONCLUSIONS

As this field is mainly focused around business and competitive practices, successful implementations and documentation of XP approaches are scarce. There are large-scale companies such as British Telecom and BMC Software that have successfully scaled agile processes. BMC Software is currently running product development teams across six locations and 11 time zones [1].

With approaches presented and evidence of existing companies, it is possible for a global company developing large-scale systems to scale to agile development processes. With the right amount of commitment to a particular process such as XP, it is possible to achieve success. The possible challenges of XP in large-scale environments should not deter one from making the change to becoming an agile development company. There are many reasonable alternative agile approaches other than XP that may scale better to one's situation. There are also different ways that one could adopt such desired methods and ways of making it work for them. As research continues and as more large-scale software development projects adopt agile methods, there should be more solutions to overcoming issues seen in scaling and better documented processes.

Acknowledgments

My thanks to Kristin Lamberty and Elena Machkasova for helping in all aspects of this paper. Thanks also to Chris Blahna, Rochelle Redding, and Andrew Hoaglund for proofreading and helping to edit this paper.

8. REFERENCES

[1] L. Barnett. Large-scale agile development. *Agile Journal*, 2006.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. `http://www.agilemanifesto.org/`, 2001.

[3] A. Cockburn and L. Williams. The costs and benefits of pair programming. In *In eXtreme Programming and Flexible Processes in Software Engineering XP2000*, pages 223–247. Addison-Wesley, 2000.

[4] K. Conboy and B. Fitzgerald. Method and developer characteristics for effective agile method tailoring: A study of xp expert opinion. *ACM Trans. Softw. Eng. Methodol.*, 20:2:1–2:30, July 2010.

[5] T. Hildenbrand, M. Geisser, T. Kude, D. Bruch, and T. Acker. Agile methodologies for distributed collaborative development of enterprise applications. In *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, pages 540 –545, march 2008.

[6] J. Patton. Ambiguous business value harms software products. *IEEE Software*, pages 50–51, 2008.

[7] A. Qumer and B. Henderson-Sellers. A framework to support the evaluation, adoption and improvement of agile methods in practice. *J. Syst. Softw.*, 81:1899–1919, November 2008.

[8] J. Shore and S. Warden. *The art of agile development*. O'Reilly, first edition, 2007.

[9] S. Soundararajan and J. Arthur. A soft-structured agile framework for larger scale systems development. In *Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the*, pages 187 –195, april 2009.

[10] Wikipedia. Agile software development — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=420668857/`, 2011. Online; accessed 28-March-2011.

[11] Wikipedia. Waterfall model — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Waterfall_model&oldid=420124129/`, 2011. Online; accessed 28-March-2011.