1-1-2012

The Impact of an Agile Methodology on Software Development Costs

Kristin Fergis
University of Pennsylvania, kfergis@wharton.upenn.edu

The Impact of an Agile Methodology on Software Development Costs

Abstract

With the emergence of the Internet, software development has become an integral part of almost every facet of business today. Because consumers have a surmounting demand for immediacy and convenience, companies are pressured to add web-based services to their product offerings. Therefore, an increasing number of resources are being allocated to the development of profitable software to meet customer needs. Because companies desire to maximize their profits, an efficient allocation of these resources is necessary to minimize costs. This can be achieved by implementing a process model that best converts their resources to quality products.

Agile software development is a relatively new framework aimed at reducing risk and production costs. It is based on iterative development and continuous feedback from all stakeholders throughout the development cycle. The switch to an agile process model from a traditional waterfall process model can reduce the risk associated with producing a large-scale software application by decreasing lead times and increasing team morale and productivity. My literature review and initial findings suggest that firms across industries can benefit from incorporating some degree of agility in their development process.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-12-09.

EAS 499 Senior Capstone Project
Project Report
April 24, 2012
Instructor: Max Mintz

The Impact of an Agile Methodology on Software Development Costs

Kristin Fergis
Advisor: Chris Murphy (cdmurphy@seas.upenn.edu)
University of Pennsylvania

ABSTRACT

With the emergence of the Internet, software development has become an integral part of almost every facet of business today. Because consumers have a surmounting demand for immediacy and convenience, companies are pressured to add web-based services to their product offerings. Therefore, an increasing number of resources are being allocated to the development of profitable software to meet customer needs. Because companies desire to maximize their profits, an efficient allocation of these resources is necessary to minimize costs. This can be achieved by implementing a process model that best converts their resources to quality products.

Agile software development is a relatively new framework aimed at reducing risk and production costs. It is based on iterative development and continuous feedback from all stakeholders throughout the development cycle. The switch to an agile process model from a traditional waterfall process model can reduce the risk associated with producing a large-scale software application by decreasing lead times and increasing team morale and productivity. My literature review and initial findings suggest that firms across industries can benefit from incorporating some degree of agility in their development process.

1. INTRODUCTION

A process model is necessary to define a timeline of deliverables for the software project and to ensure that every member of the development team, from management to engineers to consumers, understands the expectations of the workflow. While a waterfall model has been used for many years, companies are beginning to realize its inherent restrictions. Due to its linear timeline of creating requirements, implementing the complete functionality, then testing, it does not adapt well to changing consumer needs and thus exposes the project to substantial risk. My research explores the impact of switching to an agile methodology on such risk and associated costs. Agile employs an iterative development process where functionality is implemented in two-to-

four-week iterations. Therefore, all three main components of software development are repeated throughout the development cycle. This allows for change in consumer desires, as a small amount of functionality is produced every few weeks and may be easily altered. Risk is therefore less than that under the waterfall model, which would then reduce the cost of capital for software projects under the agile framework. Because this process is very adaptive and open to continuous feedback, the final product is almost perfectly aligned with customer expectations, thus resulting in high satisfaction and demand, which yield high profits. Only a sufficient implementation of an agile method, that is, a sufficient degree of agility, however, can lead to such reduced software development costs.

My research investigates the advantages and disadvantages of both process models to determine the impact of a switch in a business context. More specifically, I begin with an analysis of the general expert consensus on the limitations of both traditional and agile methods and the benefits of an agile process model. Next, I explore the effect of switching from a traditional, plan-driven model to agile methods on several measures of cost, quality, and customer satisfaction. Finally, I analyze the degree of agility, defined as the fit between the actual practices of a firm and the espoused agile practices, employed by most companies and its correlation with reduced risk and costs.

2. BACKGROUND

There have been few attempts at measuring and proving the impact of switching to an agile development process model on costs and overall software quality. Jaana Nyfjord and Mira Kajko-Mattsson from Stockholm University are in the process of integrating agile methods with risk management, while Kai Petersen and Claes Wohlin discovered various improvements by moving from a plan-driven to an incremental software development approach by performing a case study at Ericsson AB in Sweden [1][2]. However, there have been even fewer attempts to quantize the effect of such a switch.

2.1 Limitations of Traditional Methods

The most common issues associated with a traditional method include obsolete requirements, a lack of an opportunity to understand changing, current customer needs, and a high number of faults found during testing [2].

2.1.1 Requirements

Plan-driven, or traditional, methodologies rely on extensive and comprehensive planning before the implementation of any functionality can begin. Because many documented and validated requirements, which were compiled only at the first stage of the development process, must be discarded and reworked throughout the implementation and testing phases, due to changing customer needs, much time and effort is wasted outlining an exhaustive list of requirements. Therefore the fraction of implemented requirements over the total number of requirements written is small, which suggests waste. At the same time, because a plan-driven process model is not adaptive to a changing market, many of the features

that are implemented in the final product are not needed or used by customers. Thus requirements become obsolete due to long lead times (the time between project conception and implementation). Such waste increases the cost of development because those resources allocated to writing an excessive number of requirements, over those desired by current customer needs, do not produce any output of value. Those resources could have been allocated to other phases of the development process, or to another project, to obtain a positive return on investment.

2.1.2 Customer Needs

In a study comprised of 400 projects produced using a waterfall approach, customers used only a small portion of the developed application code [2]. This study confirms the lack of an opportunity to gain insight into shifting customer needs. It is also evident that there is no outlet for feedback or an opportunity to clear any misunderstandings about customer requirements within a plan-driven process model. Because the three main stages of software development (planning, implementation, testing) are completed in a linear, bulk fashion, it is increasingly difficult and expensive to pinpoint current customer needs and adjust requirements and implementation accordingly.

2.1.3 Number of Faults

Another major issue of plan-driven development is the significant number of faults found during the testing phase. Because testing is done only during the final stage of development, it is the first sacrificed when earlier phases of the process model take an unexpectedly long amount of time. In order to meet a fixed deadline, not enough testing can be done to find a sufficient amount of faults and thus the test coverage is low. Because it is usually the case that the project manager is overzealous in estimating task duration times and implementation does not elapse without a hitch, many faults persist when the product is released, which results in decreased customer satisfaction. Such low customer satisfaction can reduce repeat or future sales, which decreases profits and thus the return on investment.

Even when the planned amount of time for testing is available, it is difficult to find every fault since the entire code base needs to be examined at once. Also, since the quality of the implemented code is not known until just prior to the release date, when the functionality is tested, an unseen amount of faults may be found, which may be very expensive to fix. Thus this issue of plan-driven development also increases costs and drives down the profitability of a project.

2.2 Limitations of Agile Methods

Although agile development is not without its own issues, the number and severity of issues that impact costs and return on investment of a plan-driven methodology greatly outweigh those of an agile methodology [2]. Common issues remaining for agile after the switch from traditional methods are related to high testing lead times, low test coverage, and many teams requiring high coordination and communication from project managers. An agile process model also does not scale well to large projects, as numerous iterations are needed to complete the desired functionality and

too much time may be devoted to any single, small feature. Thus the opportunity cost of foregone production on more profitable and lean projects may be too high to employ agile methods on a large-scale project.

2.2.1 Testing Lead Times

High testing lead times can exist when a piece of functionality is only realized and implemented too late in an iteration to be tested and must wait for the next iteration to be tested. When switching to agile from a plan-driven method, such an issue is negligible compared to the excessive testing lead times of the latter, seeing as testing is saved for just prior to release (which may be several months after conception).

2.2.2 Test Coverage

Because continuous testing is needed in an agile methodology (testing at the end of every iteration), a substantive test environment containing test cases for every incremental piece of functionality is required. Agile development inherently requires that functionality be implemented in small steps with great attention to detail and current customer needs. This implies that testing must be done in small steps; it follows that such detailed implementation necessitates detailed testing. Thus more test cases are developed than in a plan-driven approach as testing is done continuously and exhaustively, which suggests that testing is the bottleneck of an agile process model [2]. Even though many test cases are developed, they tend to test on a unit level since small pieces of functionality are tested at a time. Because integration and system testing may be ignored more often than in a traditional methodology and there are simply many more test cases, testing in agile may result in low test coverage. As previously stated, low test coverage causes an increase in the number of faults found after the release of the product or more resources allocated to fixing such faults. Therefore, increased costs reduce profit and ROI.

2.2.3 Communication & Coordination

The last common issue with agile development involves management overhead. 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 to foster a sense membership and attachment to the quality of the final product. This involves much communication and collaboration between team members and management to ensure all employees across process phases (business analysts, software developers, and testers) are working together to build a successful product. Also, the project manager must make certain that the incentives of all members are aligned with the mission of the project. In addition to such team-specific attention, agile development usually employs multiple teams working on different features of a product, especially for larger projects. It follows that management must provide such quality attention to multiple teams, each with different needs and behaviors. Although this issue does not directly affect costs, the time and effort needed to manage these teams may impact the resources available to work on other projects that do not employ an agile process model.

Although agile development does exhibit some issues, they pale in

comparison to those of plan-driven development. Therefore, due to the most common issues of both methodologies, as presented by Kai Petersen and Claes Wohlin in their case study at Ericsson AB and based on my analysis of their impact on several business metrics, agile development does the least harm to potential costs and return on investment.

3. ADVANTAGES OF AGILE

Based on the literature, there are numerous benefits of agile, which I will relate to the business measures of performance that have been used thus far, costs and return on investment.

There exist many commonly perceived improvements from switching to an agile methodology. The critical improvements include more stable requirements, earlier fault detection, lower lead times for testing, increased communication, and increased adaptive capacity. I will now explain in more detail the impact of such improvements on costs and ROI and finally synthesize the overall perceived benefit of agile versus traditional methods.

3.1 Requirements

The foremost improvement from switching from a plan-driven approach is indeed related to the core competency of the approach: the planning phase of development. Agile development arose out of the need to alleviate the inherent issues of a plan-driven methodology, which are mostly related to writing requirements (many of which become obsolete) and planning. Thus it is natural to perceive how agile methods, in general, dissipate such issues.

There is much evidence to suggest that the 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. Instead of writing requirements in bulk upfront, some of which may become obsolete during subsequent phases of development, an agile process mandates iterations of no longer than four weeks where all phases are incrementally repeated. This system updates customer needs every month, at the most, when new requirements are written for each additional feature. If the requirements from the previous iteration become obsolete, which is rare since customer needs infrequently change within two to four weeks, it is easy (and cheap) to update the developed code because only a small piece of functionality was implemented. Due to such dynamic requirements, based on customer feedback, there is little need to revise existing requirements and make dramatic changes to existing functionality.

Plan-driven development does not provide an outlet for customer feedback or interaction, which is a major hindrance to productivity since consumers do not use a large portion of the developed code of the released product [2]. Agile provides much relief in this respect; customers provide feedback after almost every iteration, communicate their likes, dislikes, and new needs, and the team instantly responds to any changes. Because development is incrementally completed over time, it is not expensive to react to such changes. The sheer time and effort that is saved reworking

requirements reduces the opportunity cost of working on more profitable projects while increasing revenues from future projects that are able to begin earlier.

Also, customers receive an end product that is very aligned with their needs and are therefore willing to pay a premium for such quality, which increases revenues and ROI. This behavior has a multiplicative effect; because a quality product is released in line with demand, customer satisfaction increases, which also increases customer retention and CLV (customer lifetime value). Since customers are more likely to purchase the product due to its alignment with their needs, they associate a positive experience with the company and are incentivized to purchase future products from the same company. Thus the high present value of the revenues attributed to these relationships with various customers, or CLV, further increases ROI due to the expectation that customers will continue purchasing such products in accordance with their needs.

3.2 Fault Detection

Because testing is performed during each iteration, and not left for just prior to release, faulty pieces of code can be detected at an earlier rate than with a plan-driven process model. Each iteration includes a testing phase, which suggests that each incremental piece of code is tested with much detail. Functionality is developed in small pieces within each iteration, thus testing occurs in small steps as well; each singular feature is tested within its respective iteration. Because testing occurs continuously, faults are detected earlier and can be fixed before it increases in severity. Earlier fault detection implies many faults are found between iterations and can be cheaply fixed before much time and effort is devoted to substantial implementation, since implementation and testing occurs recursively. Since an iteration spans two to four weeks, it is not difficult to reconstruct code that contains a fault, due to the small amount of functionality that can be implemented during such a short amount of time. Also, continuous testing allows continuous testing feedback, which further improves code developed in future iterations. Because faults are detected at an earlier and continuous rate, fewer persist to release, and as previously mentioned, customer satisfaction and ROI increase.

3.3 Testing Lead Times

It follows from the previous improvement (earlier fault detection times) that agile development, in general, employs lower lead times for implementation and testing. Although it was previously stated that agile development might cause high testing lead times, it is true for only the exceptional case when a feature is implemented too late in the iteration to be tested. However, it is more often the case that those features discussed during the planning phase (of the iteration) are implemented then tested within the same iteration. Then, in general, testing lead times are greatly reduced since testing is not the ultimate action before release. Continuous testing contains the benefit of not only early fault detection, but also early and often testing, which reduces the testing lead times for pieces of functionality. Thus, testing is not sacrificed in the sake of time (if a fixed deadline must be met) due to its pervading nature, and fewer faults persist

to release. Then, for reasons previously mentioned, ROI is increased.

3.4 Communication

As a result of the strong teamwork necessary to uphold agile standards, much communication is necessary to maintain such fine relationships. Agile development not only improves communication between the company and customers but also between different employees working on the same project. Increased communication leads to increased team morale as employees begin to trust and gain the trust of their team members. A close-knit team dynamic improves productivity as members feel a part of something greater than themselves and accountable for the work of their peers. This increases team productivity and generates superior performance than the sum of all individual output. Such synergy improves the quality of the product and results in increased revenues and ROI.

3.5 Adaptive Capacity

As Philippe Kruchten implores, software development projects are not necessarily replicable to benefit from prescriptive processes [4]. An agile methodology attempts to add a dynamic component to software development; business analysts, developers, and testers must interact with each other, as well as with customers, to devise the best forward-looking strategy to meet customer demand. The adaptive nature of agile development, due to its iterative process, allows current customer needs to be incorporated into the final product, which increases demand and revenues. Finally, there exists a high correlation between process maturity and agile practices. As a project evolves into the late stages of development, firms are more likely to employ agile methods enjoy the adaptive nature of the process and facilitate the deployment of products [5].

The improvements generated by switching to an agile methodology are great in severity. The most persuasive motive to switch to agile includes the dissolution of most issues related to upfront, bulk planning. The effect on testing lead times is questionable; for the purpose of my analysis, I will assume that testing lead times is negligible relative to those under a plan-driven approach and those for other phases of the agile development cycle [3]. These improvements, in general, increase ROI, seeing as the definite reduction in the number of obsolete requirements and increase in customer satisfaction as a result of increased alignment with demand account for the majority decrease in costs and increase in revenues, respectively.

4. DEGREE OF AGILITY

Thus far I have explored the benefits of switching to an agile methodology from a traditional process model under the assumption that the software development team employs all artifacts and values of an agile process. I will now question if the degree of agility applied by a firm affects the magnitude of such benefits.

Before I explore in detail the advantages (or lack thereof) of the various tools supplied by an agile methodology (for example Daily Scrum

Meeting or Open Office Space), I would like to note that a team can only benefit from learning how to better understand and adopt agile methods in general. Maria Paasivaara and Casper Lassenius performed a case study, using an agile coaching team to instruct development teams in the use of an agile methodology, to illustrate the gains in productivity from being "more agile" [6]. They did not specify which tools are more beneficial than others, only that a more complete understanding of the agile philosophy can increase product quality and decrease overall project costs.

The most important component of an "agile mindset" is teamwork. Improving the team dynamic generates many benefits: not only would members experience a sense of belonging at their place of work, but also feel free to discuss difficult aspects of the development process with others (which there are numerous). The free flow of information and discussion between team members facilitates both team and individual growth. As individuals share experiences with others, both personal and work-related, a sense of trust is fostered throughout the team. The team then develops into a functional and productive one since all members are working together toward a common purpose. By learning from the experiences of others, individuals advance their personal growth as they cultivate such knowledge when away from the team. In order to gain these benefits in productivity (which increases ROI) from such team and individual growth, the team must embrace an agile mindset and understand that strong teamwork is very important in the success of the project.

Jaana Nyfjord and Mira Kajko-Mattsson have discovered through interviews with firms that employ agile practices that upfront, thorough planning allows for more agility in the implementation and testing phases [1]. This suggests the use of a plan-driven approach during the pre-implementation phase of development. The degree of agility that was observed in this phase depended on many factors including project type and size, criticality, degree of uncertainty, budget, and the innovative character of the project. For the majority of fixed budget projects, a more traditional approach was employed before implementation because the development team could not afford change the focus of the project based on current customer needs. At the same time, it is hard to conduct upfront, very thorough planning for small, innovative, and completely new projects. More agility in the pre-implementation phase is therefore required. The authors note that any degree of agility after performing some solid planning is acceptable as long as the project vision is not lost.

Seeing as 60% of software companies do not subscribe to only one process model, few firms adopt an agile methodology in its entirety [7]. Various fragments of agile development are utilized to help improve some failing aspect of the current process. For example, the Daily Scrum Meeting, a critical tool espoused by an agile methodology, can lead to increased collaboration, real-time information exchange, increased leadership and morale, and elevated communication. An open office space promotes faster problem solving and a reduced need for documentation but may decrease focus on work. Pair programming lends to

improved product and design quality, reduced code defects, and increased creativity. Risk is reduced per iteration as a result of timeboxing, a planning technique that divides the schedule into several separate time periods, each containing its proper deliverables, deadline, and budget. Thus many agile techniques possess great advantages and help to reduce costs, but some, such as maintaining an open office space, may detract from the project goals. To conclude, a high degree of agility is beneficial only if the selected fragments do not cause more harm than good.

5. METHODOLOGY

I will now start to explain my analysis on data collected from various individuals with experience at firms that moved to an agile methodology. I hypothesized that the switch to agile indeed reduced costs, in line with my research.

5.1 Data Collection

I developed a questionnaire, which is attached to the appendix for reference, to gauge the effectiveness of an agile method versus a more traditional approach from experienced employees. My questionnaire sought to investigate the day-to-day experiences associated with the factors of agile development that reduce cost.

Not only did I desire to gain insight about the firm's agile practices, but also about its plan-driven process before the switch. Learning about the issues the firm experienced with a more traditional approach would help to understand the appeal of agile. In addition to presenting the employees with absolute questions concerning both methodologies, I included a simple mechanism to indicate (with an arrow) an increase or decrease in critical business factors, for each development phase, as a result of the switch:

	Time	Scope	Quality	Cost
Analysis of Requirements				
Design				
Implementation				
Testing				
Maintenance				

Figure 1: Chart to Indicate an Increase or Decrease in Business Metrics After the Switch

By collecting first-hand information about the work effects of a traditional versus an agile methodology, I can better understand from real world data the benefits of a switch to agile and make a conclusion regarding the success of the process model.

5.2 Data Analysis

Introductory emails were sent to several University of Pennsylvania alumni employed at firms that recently moved to an agile process model, which include Google, Microsoft, EA Games, Bank of America, JP Morgan Chase, SAIC, Dell, and Salesforce.com. I received responses from a few alumni, after which I sent my questionnaire for completion. I believe this foot-in-door technique resulted in the highest probability of response. However, after sending out the questionnaire to those interested alumni, I received usable feedback from only one individual from Microsoft. My initial analysis was based on this piece of data.

6. RESULTS

I wished to determine whether the observed experience with an agile methodology matched my research. Indeed, Microsoft does not use an agile process throughout its entire development cycle. Also, due to resource and time constraints, some teams must stop accepting faults and begin new iterations as necessary. The alumnus stated that team members feel more tied to projects under an agile process and desire to ensure that customer needs are met. Due to these experiences, it would seem that costs are reduced as a result of an enhanced sense of belonging and strong teamwork.

At the same time, the alumnus experienced less time to react to faults later in the cycle under a traditional methodology, which also agrees with my research. Less available time to work on faults increases the number of defects found in the released product, which decreases ROI.

5. FUTURE WORK

My results did not produce any surprising points, but with time and a greater response rate from employees exposed to agile methods, more substantial results may be found. In the future, much more data may be collected to generate more extensive analyses. The increases or decreases in costs and return on investment may also be quantized to provide more quantitative evidence of the benefits of agile development.

7. REFERENCES

[1] Nyfjord, Jaana, and Mira Kajko-Mattsson. *Degree of Agility in Pre-implementation Process Phases*. Rep. Berlin: Springer-Verlag, 2008. Print.

[2] Petersen, Kai. *The Effect of Moving from a Plan-Driven to an Incremental Software Development Approach with Agile Practices*. Rep. Print.

[3] Petersen, Kai. *An Empirical Study of Lead-Times in Incremental and Agile Software Development*. Rep. Springer, 2010. Print.

[4] Kruchten, Philippe. *A Plea for Lean Software Process Models*. Rep. 2011. Print.

[5] Ronkko, Mikko, Antero Jarvi, and Markus M. Makela. *Measuring and Comparing the Adoption of Software Process Practices in the Software Product Industry*. Rep. Berlin: Springer, 2008. Print.

[6] Paasivaara, Maria. *How Does an Agile Coaching Team Work?: A Case Study*. Rep. New York: ACM, 2011. Print.

[7] Paasivaara, Maria. *How Does an Agile Coaching Team Work?: A Case Study*. Rep. New York: ACM, 2011. Print.

APPENDIX

QUESTIONNAIRE

The Switch to Agile Development

A: Background Information

1. What is your job title?

2. What is your firm name?

3. How long have you been with your firm?

4. How would you best describe the industry in which your firm operates?

5. Approximately how many people work in software development?

B: Agile Methodology

6. Why did your firm start using agile methods for software development?

7. Which agile process does your firm utilize (Scrum, XP)?

8. To what extent does your firm follow the procedures and principles set by this process?

9. What is the average project lifetime, from inception to a functional system, under agile development?

10. Do developers and testers feel more tied to projects (IE do they care about the projects' impact on the overall firm)?

11. How much time and resources do you spend fixing bugs?

12. Do customers feel more satisfied with your end product?

C: Traditional Methodology

13. What software development methodology did you employ before agile?

14. What did you not like about this methodology?

15. What was the average project lifetime, from inception to a functional system, under this methodology?

16. How much time and resources did you spend fixing bugs?

17. Overall, do you prefer agile methods to this methodology?

18. If so, why?

Please indicate below which of the following factors have increased or decreased (mark with an arrow), for each development phase, as a result of switching to agile methods:

- Time: the time to completion
- Scope: the number of features your firm is able to include or implement
- Quality: the number of bugs (internal) or customer satisfaction (external)
- Cost: the amount of resources (monetary or otherwise) allocated

	Time	Scope	Quality	Cost
Analysis of Requirements				
Design				
Implementation				
Testing				
Maintenance				