Cost of Change in Agile Development Environments

Matthew J. Martin
North Carolina State University
Raleigh, NC, USA
Email: matt@matthewmartin.me

***Abstract*—Fill this in later.**

I. Introduction

The science of improving software development methods is not one that has traditionally attracted attention or extensive research. The Waterfall method of creating software has been the standard for decades. This strategy was characterized by long update cycles and extensive planning early in the process. A classic example of this is Microsoft's ”Service Pack” update scheme for its Windows operating system. Windows users would typically wait anywhere from six to 12 months to see an update in their software. These updates would be massive and take hours to complete.

This standard frustrated users and developers alike, but no viable alternative was proposed until 2001, when Kent Beck and 16 others formalized a new methodology concerning software development called Agile. Agile is the antithesis of Waterfall, emphasizing responsiveness and frequency of change. One of the biggest stated benefits of Agile over Waterfall is the improvement in cost of change in Agile projects. Cost of change is a measurement of the resources necessary to make a change to a product while it is still being developed.

In traditional development environments, the cost of change curve grows exponentially as time goes on. That is, the later in the development process that a change occurs, the more expensive it will be. One of the biggest attributes of Agile development is that the cost of change theoretically levels out as time goes on, adopting more of a *log(n)* curve, as can be seen in Figure 1 [4]. Therefore, it is beneficial to put off making bigger decisions to later in development because the increase in cost to change will be negligible, and there is a higher likelihood of making the correct decision later in the process [3].

However, the validity of this claim is uncertain. No exhaustive study exists that thoroughly proves the legitimacy of this assertion. This is important to investigate because it is not just one premise of the Agile methodology, it is, as Kent Beck puts it, ”*the* technical premise of XP [Extreme Programming],” [3] which is one of the most common forms of Agile development. If this hypothesis were to be disproven, then many of the merits of Agile development and Extreme Programming would be invalidated. The software development industry will be uniquely interested in this result, as, after all, it is a business, and businesses value ways to save money. If it is shown that Agile development is not the silver bullet that the founders of the methodology marketed it to be, its popularity will begin to decrease rapidly.

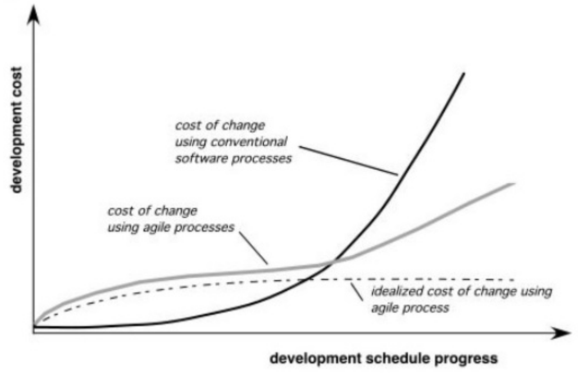

Fig. 1: Traditional versus Agile cost of change curves

This paper seeks to investigate the true nature of cost of change in Agile environments. First, related work will be examined to provide an overview of the study of cost of change in different environments. Then data (from Tom? from all sorts of companies?) regarding cost of change will be extracted and compared to the claimed cost of change curve for Agile. Finally, possible threats to validity and avenues of future research will be discussed.

The following research questions were used to study this problem, and will be referred to throughout the remainder of this paper:

- **RQ1**: For Agile projects, does cost of change actually reflect the curve suggested by the creators of Agile?
- **RQ2**: Does the size of the development team or project have any effect on this?

II. Background

A. The Agile Methodology

The enthusiasm of project managers and software developers for Agile development has been remarkable to say the least. Ever since Kent Beck's book *Extreme Programming Explained* [3] was published in 2000, adoption of Agile methodologies in the software industry has grown dramatically. According to the 10th Annual State of Agile Report [17], 43% of companies surveyed reported that a majority of their projects used Agile methodologies, and only 4% reported no use of Agile whatsoever.

The Agile Manifesto was written in 2001 by 17 pioneers of the methodology. The Manifesto contains the following four points [4]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

These principles encapsulate the essence all of the different practices of Agile, like Scrum and Extreme Programming, which are the two most popular implementations of Agile [17].

The Agile methodology claims many advantages over traditional development styles. According to Schmidt et. al. [13], these benefits include better overall software quality, improved feature scope, and a better sense of work pride through teamwork. Some other benefits of this process can be seen through the definition of ”agility” found in the Agile Manifesto [4]:

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

These factors all yield ”rapid, incremental delivery of software” [4]. However, there are some who say Agile development is not as beneficial as its founders claim.

Dan Turk, Bernhard Rumpe, and Robert France identify six limitations of Agile development [16]. Two of these are of particular interest concerning **RQ2**:

- Limited support for development involving large teams
- Limited support for developing large, complex software

Other authors [14], [10] also cite large project size as a reason to stay away from Agile, instead opting for traditional approaches. Both of these drawbacks of Agile development lend credence to my suspicion that the success of Agile may in part depend on the size of the team or project.

The arguments for and against Agile development in different scenarios are seemingly endless. The scope of this paper, however, will remain within an analysis of cost of change in Agile projects, to see if that claimed advantage is tenable.

B. Motivation

The issue of cost of change is important to the Agile versus Waterfall debate because cost of change is so central to Agile's methodology [3]. It is the primary selling point, and one of the main reasons that Agile software development has exploded in popularity over the past several years.

But it is important that companies wishing to adopt Agile practices have an accurate and empirical reason for why it will cost less than traditional development approaches. This paper will eliminate the mystery behind Beck's cost of change curve (Fig. 1), and provide a definitive answer to those who question Agile's effectiveness in minimizing cost of change.

C. Related Work

Some research has been made into general methods of estimating cost of change for software development projects [2], [5], [8], and many have begun to apply cost estimation to Agile development projects [18], [9], [15], [7].

Barry Boehm et. al. [5] explore some of the most common methods of cost estimation in software engineering. These include the *Doty model* and the popular *COCOMO model*, as well as newer algorithms like *The Tausworthe Deep Space Network model* and the *Jenssen model*. He then outlines areas in need of more research to make improved cost estimation models including software size estimation, software project dynamics, and software complexity metrics.

Bakota et. al. [2] investigate the relationship between software entropy and cost. They use maintainability to measure the entropy of software, and found that maintainability decreases over time (entropy increases). Furthermore, maintainability was in an exponential relationship with development cost, and they could accurately predict future cost with their model.

Hanssen et. al. [8] explore the connection between software entropy and Agile projects. They found that the rapid change found in Agile projects exacerbated the growth of entropy throughout the development process. Connecting this conclusion to Bakota's conclusion, an argument could be supported that links Agile development to higher cost of change via Agile's effect on software entropy.

Garg and Gupta [7] propose a new cost estimation technique specific to Agile projects using Principal Component Analysis (PCA). They use PCA to identify the attributes that contribute most to the development cost, and then give a total cost estimation based on that information. They further tune this to Agile projects by applying various constraints to the extracted features. Interestingly, they found that Team Size and Project Complexity were the second and fourth most influential factors in estimating cost, respectively, which relates to **RQ2**.

Kang et. al. [9] propose another model specific to estimating the cost of Agile projects. They observe that traditionally, Agile teams estimate cost based on ”story points,” which are essentially features of the project. However, they observe that this is a subjective process and can be inconsistent. They propose breaking down story points into individual function points, where each function point is a narrow task that is part of a story point. Function points are easier and truer metric by which to estimate cost. They also account for changes throughout the process by using a *Kalman Filter* algorithm to recalculate the project's velocity on a daily basis, and revise the total cost estimation.

Ziauddin and Zia [18] note that many cost estimation models used by Agile teams rely on a relative approximation of effort. Their model is a more quantitative approach and takes four inputs: Story Size Metric, Story Complexity Metric, Friction Factor Metric, and Variable Factor Metric. They provide detailed descriptions of what a particular value for any of these inputs should indicate, and their model outputs time and total cost for the project.

III. Experimental Data

A. Background

The data used in this study is from two large, industry leading companies. Company A provided information regarding several open source projects that they contribute to, and Company B offered data about several of their proprietary projects, including some that were more Agile and some that were more traditional in their development styles.

The data from Company A's open source projects included primarily information regarding commits and issue reports. It contained all information pertaining to any given commit, such as the time of the commit, the committer, any issue reports associated with the commit, and how many lines of code that the commit affected. This information will all be useful in measuring cost to change because the commits can be treated as changes to the project, and the cost or effort of the commit can be estimated from the number of lines changed, files changed, or some other surrogate. However no information regarding the "agility" of these projects is explicitly available, so a method of measuring agility was employed to rank these projects in terms of how much or how little they adhere to Agile methodologies [12], [6], [1].

The data from Company B provided even more information specific to the research questions presented in this paper. These projects, at their inception, chose a methodology to follow from a set of templates that the company provided. The templates ranged from traditional waterfall style development to Agile development. This information will be extremely useful in comparing cost to change between these different projects that use different methodologies. Within each project, they have data divided into WorkItems, which are essentially individual tasks, features, or user stories that need to be integrated into the project. Within each WorkItem, there is information regarding how much time a developer spent on it, how many developers worked on it, and a lot of other useful data. All of this can be used to make an accurate calculation of the cost of any given change.

B. Pre-Processing and Analysis

1) Company A: Company A provided data from numerous open source projects to which they contribute. The data is in a collection of JSON files within each project directory. One of these files in each directory joins all of the data pertaining to a commit: the issue associated with the commit (if there was one), the commit time, the magnitude of the commit, and other information regarding it. This is the file that will be most useful. We will profile the data in accordance with the criteria suggested by Petersen and Wohlin in [11], focusing on Project Kafka.

First, the *product* will be described. Though the first commit to Kafka was made in July 2011, consistent development on the project did not begin until November 2012. This makes the project almost five years old. Kafka is a fairly large project, with over 2,500 commits and 198 contributors. It is built mostly in Java, followed by Scala and Python.

The development *process* is also important to note. The work-flow of the project is as follows: jira to track and post issues, formal improvement proposals to plan major changes, and GitHub pull requests to review and merge changes.

The types of *people* developing Kafka also provide insight. As an open source project, anyone can contribute. This makes the experience of contributors questionable, but extensive pull review and testing account for any quality gap between developers with different levels of experience. However, of the 198 contributors, only 30 have committed over 10 times and only 15 have committed over 40 times, showing that the majority of development of Kafka is done by a few core contributors.

The cost of a commit (a change) needs to be calculated from this information. These are the data points that we propose will have an impact on the cost of any given change:

- Time taken between issue creation and commit, t
- Experience of the committer, e
- The size of the commit, s
  - Number of lines of code in the commit, l
  - Number of files affected by the commit, f
- Number of commits for a committer, c
- Priority of the issue, p

An issue with the data is that many of the commits (approximately half) do not have an issue associated with them. Therefore it is impossible to accurately estimate the time it took to make the commit, since there is only a timestamp for when the commit was made, not one for when work began on the commit. These commits will be omitted from the analysis.

Using the cost features identified above, Equation 1 was created as a calculation of cost.

$$cost = aMathematicalCombinationOf : t, e, s, c, p \quad (1)$$

When beginning to analyze this data, I first thought it could be interesting to see the breakdown of issue types among all issues reported. Figure 2 shows this distribution for four of the projects. This figure shows that the four projects sampled all have similarly proportioned issue types. This information can be useful since the type of issue may have a relationship to the cost to fix that issue.

2) Company B: The data from Company B is in a different format, but contains much of the same information as Company A's data. Company B provided data from three projects that were developed using Agile methodologies and three projects that were developed with more traditional styles. For each project, information regarding the progress of every change made to the project was provided. This included priorities of each WorkItem and timestamps for when the WorkItem was created and when it was resolved. I will be able to analyze this in a similar manner to how I analyze Company A's data since it is essentially the same information about the projects.

References

[1] BM Arteta and RE Giachetti. A measure of agility as the complexity of the enterprise system. *Robotics and Computer-Integrated Manufacturing*, 20(6):495–503, 2004.

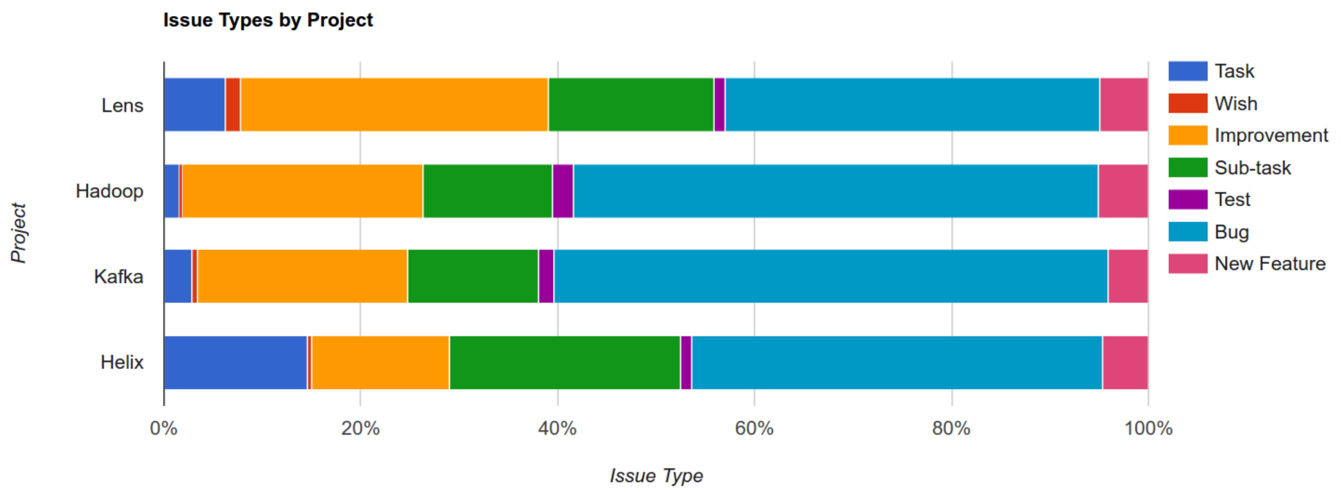

Fig. 2: Issue distributions for different open source projects from Company A

[2] T. Bakota, P. Hegeds, G. Ladnyi, P. Krtvlyesi, R. Ferenc, and T. Gyimthy. A cost model based on software maintainability. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 316–325, Sept 2012.

[3] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[4] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[5] Barry W Boehm et al. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981.

[6] Taghi Javdani Gandomani and Mina Ziaei Nafchi. Agility assessment model to measure agility degree of agile software companies. *Indian Journal of Science and Technology*, 7(7):955–959, 2014.

[7] S. Garg and D. Gupta. Pca based cost estimation model for agile software development projects. In *Industrial Engineering and Operations Management (IEOM), 2015 International Conference on*, pages 1–7, March 2015.

[8] G. Hanssen, A. F. Yamashita, R. Conradi, and L. Moonen. Software entropy in agile product evolution. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10, Jan 2010.

[9] S. Kang, O. Choi, and J. Baik. Model-based dynamic cost estimation and tracking method for agile software development. In *Computer and Information Science (ICIS), 2010 IEEE/ACIS 9th International Conference on*, pages 743–748, Aug 2010.

[10] Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham, and Soo Fun Tan. Software development life cycle agile vs traditional approaches. In *International Conference on Information and Network Technology*, volume 37, pages 162–167, 2012.

[11] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 401–404. IEEE Computer Society, 2009.

[12] Asif Qumer and Brian Henderson-Sellers. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and software technology*, 50(4):280–295, 2008.

[13] Christoph Tobias Schmidt, Srinivasa Ganesha Venkatesha, and Juergen Heymann. Empirical insights into the perceived benefits of agile software engineering practices: A case study from sap. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 84–92. ACM, 2014.

[14] Marian Stoica, Marinela Mircea, and Bogdan Ghilic-Micu. Software development: Agile vs. traditional. *Informatica Economica*, 17(4):64, 2013.

[15] Binish Tanveer, Liliana Guzmán, and Ulf Martin Engel. Understanding and improving effort estimation in agile software development: An industrial case study. In *Proceedings of the International Workshop on Software and Systems Process*, ICSSP ’16, pages 41–50, New York, NY, USA, 2016. ACM.

[16] Dan Turk, Robert B. France, and Bernhard Rumpe. Limitations of agile software processes. *CoRR*, abs/1409.6600, 2014.

[17] VersionOne. The 10th annual state of agile report. https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf, 2015.

[18] Shahid Kamal Tipu Ziauddin and Shahrukh Zia. An effort estimation model for agile software development. *Advances in Computer Science and its Applications (ACSA)*, 314:314–324, 2012.