Comparing Extreme Programming and Waterfall Project Results

Feng Ji
Carnegie Mellon University
Silicon Valley Campus
Mountain View, CA, 94035
jojojifeng@gmail.com

Todd Sedano
Carnegie Mellon University
Silicon Valley Campus
Mountain View, CA, 94035
todd.sedano@sv.cmu.edu

Abstract

Waterfall and Extreme Programming are two software project methods used for project management. Although there are a number of opinions comparing the two methods regarding how they should be applied, none have used project data to clearly conclude which one is better. In this paper, we present the results of a controlled empirical study conducted at Carnegie Mellon University in Silicon Valley to learn about the effective transition from traditional development to agile development. We conducted a comparison research against these two approaches. Multiple teams were assigned a project; some used Waterfall development, others used Extreme Programming. The purpose of this research is to look at advantages and disadvantages based upon the outcomes, generated artifacts, and metrics produced by the teams.

1. Introduction

1.1. Agile vs Traditional

Since the early 1970s, numerous software managers have explored different ways of software development methods (such as Waterfall model, evolutionary model, spiral model etc.) those have been developed to accomplish these goals and have been widely used by the software industry [1]. Methodologists often describe the Waterfall method as a stereotypical traditional method whereas they describe Extreme Programming as the stereotypical agile method. The Waterfall model, as the oldest traditional software development method, was cited by Winston W. Royce in 1970 [2]. He divided the software development lifecycle into seven sequential and linear stages: Conception, Initiation, Analysis, Design, Construction, Testing, and Maintenance. The Waterfall model is especially used for large and complex engineering projects. Waterfall's lasting impression upon software engineering is seen even in the Guide to Software Engineering Body of Knowledge which introduces the first five knowledge areas based upon their sequence in the Waterfall lifecycle even though the Guide does not recommend any particular lifecycle [3].

Although the Waterfall model has been adopted in many large and complex projects, it still has some inherent drawbacks, like inflexibility in the face of changing requirements [1]. If large amounts of project resources have been invested in requirements and design activities, then changes can be very costly later. High ceremony documentation is not necessary in all projects. Agile methods deal well with unstable and volatile requirements by using a number of techniques of which most notable are: low ceremony documents, short iterations, early testing, and customer collaboration. Kent Beck and Cynthia Andres define Extreme Programming 2.0 with many practices [4], like Pair Programming, Test First Programming, and Continuous Integration and so on. These characteristics enable agile methods to obtain the smallest workable piece of functionality to deliver business value early and continually improving it while adding further functionality throughout the life of the project [5].

1.2. PET project background

Carnegie Mellon University Silicon Valley students start their masters program with the Foundations of Software Engineering course. This course is team-based, project-based, and mentored. Each team builds The Process Enactment Tool (PET). The user personas are software developers and managers. The tool helps users plan, estimate, and execute a project plan while analyzing historical data. The tool's domain encourages students to learn about software lifecycles and methods while understanding the benefit of metrics and reflection.

1.2.1. PET 1.0: In 2001, Carnegie Mellon had one of the largest outsourcing firms in the world develop Pet 1.0. Later the student teams were brought in to do the next release. The initial offerings of the course had the teams follow a Waterfall lifecycle. The faculty decided to use Extreme Programming as the method for the Foundations course because it was an agile method, it had good engineering practices, and it was a safe sandbox environment for engineers to try paired programming since many managers in industry were initially skeptical about its benefits. In 2005, the faculty allowed three of the sixteen teams tried our new curriculum to see if there were any serious issues in the switch, while other thirteen teams continued to follow a start point in 2004. The feedback was extremely positive so in 2006, all teams followed Extreme Programming. For the project plan duration, Waterfall teams needed fifteen weeks to finish their tasks where as Extreme Programming teams were given only thirteen weeks, a 13% reduction in time.

1.2.2. PET 1.1: In 2005, the VP of Engineering advised the three teams that rewriting the code from scratch would be easier than working with the existing code base. Team 30:1 decided to use the latest in Java technologies including Swing and Hibernate. PET 1.1, the team's product became the starting point for the students in the following year.

1.2.3. PET 1.2: In 2008, the faculty switched the core technology from Java to Ruby on Rails. Ruby on Rails' convention over configuration, afforded a lower learning curve for students. For Pet 1.2, students would build their projects from scratch.

2. Related work

Much research has been done as to when to use an agile method and when to use a traditional method. For example, Boehm Turner's home grounds look at several characteristics, criticality, culture, and dynamism [6]. Our paper aims to extend these limitations to some degree by estimating Waterfall and XP in an academic case study, which provide a substantive ground for researchers before replicating their ideas in industry.

Basili [7] presented a framework for analyzing most of the experimental work performed in software engineering. We learned that how to conduct a controlled experiment. Andrew and Nachiappan [8] reported on the results of an empirical study conducted at Microsoft by using an anonymous web-based survey. They found that one third of the study respondents use Agile methodologies to varying degrees and most view it favorably due to improved communication between team members, quick releases and the increased flexibility of agile designs. Their findings that we will consider in our future work is that developers are most worried about scaling Agile to larger projects, and coordinating agile and traditional teams. Our work is closely related to the work by Ming Huo et al [9]. They compared the Waterfall model with agile processes to show how agile methods achieve software quality. They also showed how agile methods attain quality under time pressure and in an unstable requirements environment. They presented a detailed Waterfall model showing its software quality support processes. Other work has only illustrates one or some Agile practices such as pair programming [10].

3. Experimental methodology

Our research was conducted primarily using Glaser's steps [11] in the constant comparison method of analysis. Step1: Begin collecting data. We collected more than 50 teams' detailed data during a five year period as Table 1 shows.

Table 1. Team building the same project

	2004	2005	2005	2006	2007	2008
Method	Waterfall	Waterfall	XP	XP	XP	XP
Language	Java	Java	Java	Java	Java	Ruby
Project	PET1.0	PET1.0	PET1.0	PET1.1	PET1.1	PET1.2
Numbers of Teams	10	13	3	9	6	11

Step2: Look for key issues, recurrent events, or activities in the data that become categories for focus. The approach in software design makes us categorize the data into two distinctive software development methods, namely Waterfall and Extreme Programming.

Step3: Collect data that provides many incidents of the categories of focus with an eye to seeing the diversity of the dimensions under the categories. According to Basili[7], we provided some metrics to compare these two categories, Waterfall and XP.

Requirements Metrics

M1: Numbers of UI screens (ie. mockup) M2: Numbers of use cases (story cards)

M3: Pages of Software Requirements Specification (SRS) documents

M4: Pages of User Requirements Documents (URD)

Design Metric

M5: Pages of detailed design documents

Implementation Metrics

M6: Lines of code M7: Percentage of lines of comments to lines of source code

M8: Lines of test cases M9: Ratio of lines of test code to lines of program code

Step4: Write about the categories that we are exploring, attempting to describe and account for all the incidents we have in our data while continually searching for new incidents.

Step5: Work with the data and emerging model to discover basic social processes and relationships.

Step6: Engage in sampling, coding, and writing as the analysis focuses on the core categories. During 2005, there were 13 teams following Waterfall and 3 teams following XP during the same period of time. **These three teams, team Absorb, GT11 and 30:1 are interesting teams to examine as we can compare their data against the Waterfall teams doing the exact same project.**

4. Experimental results

4.1. UI screens (M1) and Story cards (M2) comparison

These wide ranges can be seen in Table 2 and Table 3 where the standard deviation of the UI mockups is often half the document size. Comparing use cases to story cards in Table 3, we see that the standard deviation for use cases is much lower than the standard deviation for story cards. This is expected since use cases are a higher ceremony document when compared to story cards. Teams might give little consideration to how to represent each feature on a story card whereas a team writing a use case step by step how a user will use the system will spend much more time thinking about the coupling and cohesion of each use case.

Table 2. Average numbers and Standard Deviation of mockups

Year	2004	2005	Absorb	GT11	30:1	2006	2007	2008
Average mockups	15.5	11.8	17	18	9	15	12.8	17.7
Standard Deviation of mockups	6.6	6.3				5.4	3.1	8.8

Table 3. Average numbers and Standard Deviation of use cases/story cards

Year	2004	2005	Absorb	GT11	30:1	2006	2007	2008
	User cases	User cases	Story cards	Story cards	Story cards	Story cards	Story cards	Story cards
Average Number	18.7	18.9	15	13	18	16.6	18.3	16.6
Standard Deviation	2.3	1.6				7.5	6.8	8.0

4.2. Requirement documents (M3&M4)

Starting with PET 1.0, Waterfall teams on average add 1.7 use cases and modified 2.0 use cases. Teams were given a 28 page System Requirements Specification (SRS) and on averaged finished with a 34 page SRS. XP teams starting with PET 1.0 were given the same starting documents. Instead of modifying them, the teams created story cards that represented each new feature. **Instead of spending time on writing use cases, XP teams started coding sooner. Because XP has an emphasis on low ceremony documents, they had more time to code resulting in an effort savings for the teams.**

4.3. Comparing the size of the detail design documents (M5)

There are some insights from Table 4. Waterfall teams using Pet 1.0 started with a 21 page Detailed Design Document (DDD), which they altered to reflect their new use cases. Waterfall teams typically did not update their design documents at the end of the project. Given the scope of the project, Waterfall teams' final code matched the original design with respect to new classes.

Table 4. Average pages and Standard Deviation of Detail Design Documents

Year	2004	2005	Absorb	GT11	30:1	2006	2007	2008
Starting Point	21	21	21	21	0	14	14	0
Average DDD	25.8	31.1	18	22	14	18.3	12.5	9.5
Standard Deviation	8.39	7.48				7.70	7.48	5.19

XP teams increased their design documents with each iteration. Because the XP teams followed Test-Driven Development, they wrote their code and had an emergent design. At the end of each iteration, the teams were asked to update the design document to reflect important design decisions they had made during that iteration. Therefore, **the design document serves a different purpose in XP.** It is not a template or blueprint for future construction. Instead, it can be a guide for understanding why certain decisions were made. In this regard, it is a biography of the development, not a plan of action.

4.4. New lines of source code and comments, Percentage of comments in codes

Table 5 shows that Waterfall teams starting with Pet 1.0 produced lines of code with a wide variance. The two XP teams starting with Pet 1.0 fell right within the middle of the average. Because instead of producing some documents up front, the XP teams spent a longer time coding, one would expect them to produce more lines of code. The research results also show that XP Teams had a higher percentage of comments in source code.

Table 5. Average and Standard Deviation of new lines in code

Year	2004	2005	Absorb	GT11	30:1	2006	2007	2008
Language	Java	Java	Java	Java	Java	Java	Java	Ruby
Average new lines in code	9,429	11,910	13,288	14,689	0	9,628	8,572	3,670
Standard Deviation	7,946	9,851				4,920	5,465	1,507
Lines of test codes	3378	4164	1380	3186	947	3555	2212	3,255
Ratio of test codes to program code	8%	13%	4%	8%	8%	16%	10%	90%

4.5. Submitted lines of test codes and ratio of test code to program code

The observation of these two metrics in Table 5 shows that the amount of test code written by the Waterfall teams equals the amount of test code written by the XP teams. **Initially the faculty thought that Test-Driven Development would increase the amount of testing code, however, given a slow adoption rate of Test-Driven Development, programmers resorted to what was familiar and thus produced similar results.**

5. Conclusion

In this paper, we observed and presented the data from five years of 50 teams developing the same project each year and the affects of transitioning from Waterfall to Extreme Programming. The characteristics between these two methods were evaluated and compared. Waterfall teams spent more time creating high ceremony documents where as Extreme Programming teams spent more time writing code and documenting their design in their code. Surprisingly, the amount of code and features completed were roughly the same for both methods suggesting that on a three month project with three to four developers it doesn't matter the method used. It is challenging to conduct this kind of analysis of the data in hindsight. Given that this is not a toy problem, and the freedom teams have in the execution of their projects, setting up this kind of experiment properly in advance is also challenging.

6. References

[1] Sommerville, Software engineering, 8th ed., New York: Addison-Wesley, Harlow, England, 2006.

[2] W.Royce, Managing the Development of Large Software Systems, IEEE WESTCON, Los Angeles, 1970.

[3] A. Abran and J. W. Moore, Guide to the software engineering body of knowledge: trial version (version 0.95) IEEE Computer Society Press, Los Alamitos, CA, USA, 2001.

[4] Kent Beck and Cynthia Andres, Extreme programming eXplained: embrace change, Second Edition, MA: Addison-Wesley, 2004.

[5] Mike Cohn, Agile estimating and planning, Prentice Hall Professional Technical Reference, Nov 11, 2005.

[6] Barry, Boehm and Richard Turner, Balancing Agility and Discipline: A Guide for the Perplexed, Addison Wesley, August 15, 2003.

[7] Basil, V.R., Selby, R. and Hutchens, D., Experimentation in Software Engineering, IEEE Transactions on Software Engineering (invited paper), July 1986.

[8] Andrew Begel and Nachiappan Nagappan, Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study, MiIEEE Computer Society MSR-TR-2007-09, no. (2007): 10.

[9] Ming Huo, June Verner, Muhammad Ali Babar, and Liming Zhu, How does agility ensure quality?, IEEE Seminar Digests 2004, (2004):36.

[10] Jan Chong, Robert Plummer, Larry Leifer, Scott R. Klemmer, and George Toye. Pair Programming: When and Why it Works, In Proceedings of Psychology of Programming Interest Group 2005 Workshop, Brighton, UK, June 2005.

[11] Glaser, Barney G, Strauss, and Anselm L., The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine Publishing Company, Chicago, 1967.