JOURNAL OF INFORMATION SYSTEMS APPLIED RESEARCH

Volume 13, Issue. 1
March 2020
ISSN: 1946-1836

In this issue:

iscap
Information Systems & Computing Academic Professionals

The **Journal of Information Systems Applied Research** (JISAR) is a double-blind peer reviewed academic journal published by ISCAP, Information Systems and Computing Academic Professionals. Publishing frequency is three issues a year. The first date of publication was December 1, 2008.

JISAR is published online (http://jisar.org) in connection with CONISAR, the Conference on Information Systems Applied Research, which is also double-blind peer reviewed. Our sister publication, the Proceedings of CONISAR, features all papers, panels, workshops, and presentations from the conference. (http://conisar.org)

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the conference. At that point papers are divided into award papers (top 15%), other journal papers (top 30%), unsettled papers, and non-journal papers. The unsettled papers are subjected to a second round of blind peer review to establish whether they will be accepted to the journal or not. Those papers that are deemed of sufficient quality are accepted for publication in the JISAR journal. Currently the target acceptance rate for the journal is about 40%.

Questions should be addressed to the editor at editor@jisar.org or the publisher at publisher@jisar.org. Special thanks to members of EDSIG who perform the editorial and review processes for JISAR.

JOURNAL OF INFORMATION SYSTEMS APPLIED RESEARCH

An Empirical Study of Post-Production Software Code Quality When Employing the Agile Rapid Delivery Methodology

Laura F. Poe
laurapoe@verizon.net
Longwood University
Farmville, VA

Elaine Seeman
seemane@ecu.edu
East Carolina University
Greenville, NC

Abstract

In response to the business need to adopt a faster delivery model to enable them to stay ahead in the marketplace, organizations implementing Agile practices expect to deliver projects faster and with higher quality. Widespread assumptions of increased code quality for software implementations using Agile require empirical investigation. The purpose of this paper is to evaluate software delivery with an emphasis on the quality of the software code. The outcome of this research will assist business leaders with making informed decisions on selecting a successful project methodology. While numerous factors can impact project delivery, this case study of DigiTek LLC evaluates their software development project teams' software delivery hours to the number of defects encountered during development and after implementation to production. Teams using the traditional Waterfall methodology had slightly higher production code quality when compared to teams using the Agile methodology across similar software development products. Companies planning to adopt Agile should evaluate the impacts to code quality and consider other factors as part of the decision to transition.

Keywords: Software engineering; case study; Agile; code quality; rapid delivery; Waterfall

1. INTRODUCTION

This paper compares Agile and Waterfall project methodologies and the quality of software code based on the number of production defects relative to software development hours. The adoption of the Agile methodology in software development projects has been considered a means to stay ahead of technology trends that are sweeping the industry. Agile is a methodological response to the rigid requirements and design processes of earlier methodologies that often lead to fixed project scope, significant modifications to design late in the software development life cycle, and customer dissatisfaction. Success has been primarily determined by the delivery speed and the ability to change design and scope based on customer input. Additionally, the quality of the software code should be analyzed to determine its software reliability in conjunction with rapid delivery. A quantitative, empirical analysis is necessary to determine the effect of Agile practices on the quality of code being implemented. This study was performed to evaluate Agile's impact on code quality by directly comparing the Waterfall defect rate to the defect rate using Agile. The benefit of the case study is the ability to obtain valid project results in a live industrial setting.

The emergent Agile methodology professed as delivering higher quality products demands a comparative research study to evaluate the accuracy of the claims and to classify characteristics leading to higher quality software code. While numerous factors can determine team performance, such as team diversity of skill set as described in Lee and Xia's study on Software development agility (2010), methodological success and team performance can be measured based on the quality of the product delivered. This paper discusses the process of conducting a quantitative study with eight teams employed at DigiTek LLC in order to obtain actual measurements of production code quality after release by teams using both Waterfall and Agile methodologies. The advantage of an empirical study is the delivery of reliable results that can be used to compare and contrast industry averages. At this juncture, few empirical studies have been performed that provide a side-by-side analysis between Waterfall and Agile projects for code quality impacts using a multidimensional construct. The metrics used provide consistent and equal factors for statistical comparisons.

We intend to provide the following contributions on agile literature. The results of the study will be helpful to practitioners, who are seeking faster software delivery releases, to choose the appropriate project methodology for their organization dependent upon complexity of software development team structures.

The sections of this paper are divided as follows: the fundamental differences between Waterfall and Agile and highlights the main reasons organizations choose the Agile methodology; issues related to Agile and software testing; the scope of the case study and research questions; and the results of the case study with recommendations.

2. THEORETICAL BACKGROUND

Waterfall and Agile Approaches

The emergence of the Agile methodology in the 1990's went largely undetected by the software development industry. Few companies pursued the methodology to deliver new functionality or code enhancements. The shift towards rapid delivery methodologies over the last decade demanded that the software development community question the efficiency of traditional software processes and models (Sampaio, Vasconcelos, & Sampaio, 2004).

Competition in the technology sector was a distinct motivator for companies and led to market driven changes in methodology to accommodate the delivery of technological advancements at near record paces. Agile organizations have the ability to react swiftly and decisively to sudden shifts in overall market conditions, such as the emergence of new competitors and new technologies (Holbeche, 2015). In an effort to deliver products more quickly, organizations evaluate optimizations to internal processes, and as a result, shift from the traditional Waterfall project methodology to Agile. The shift to Agile requires changes to project team member roles and transitions software development from the step by step approach to the combined step of design, development, and testing. Typically correlated with IT and software development, Agile is being implemented as a model across organizations as a means for tracking and delivering work.

The Waterfall method associated with the Software Development Life Cycle had been uncontested for years until the rise of Agile. IT projects using Waterfall operated in a backwards scheduling approach by selecting a go-live, implementation date, before the project began. Once the date was determined, the project manager and team would identify the tasks required to implement the project, allocating time for each of the activities. The Waterfall methodology requires that each phase of the project reach a stage of completion before the next phase begins. Agile practices allow the team to decide how much work can be accomplished per iteration before beginning the work. These distinctly differing models of delivery management maintain the same end goal of releasing a quality product.

IT managers often cite the reduction of time spent in initial planning, leading to an evolving and more efficient process as an advantage of Agile (Dyba et al. 2008). Evidence suggests that the rigid development processes of the Waterfall methodology results in rework, customer dissatisfaction (Dybe et al. 2008), and missed implementation dates. Moreover, project team members often endure increased working hours necessary in order to meet the go-live dates. In addition, date-driven projects tend to go over budget due to increased resources and project hours for a successful implementation.

Methodological Distinctions

Simple, key methodological distinctions between the Agile and Waterfall methodologies (Table 1) can be summarized in the overall management of

the project. Practitioners utilizing the Waterfall methodology are accustomed to intense up-front project planning. With the Agile methodology, the focus moves to a more independent model with self-managing teams. A hybrid approach is usually appropriate for project planning to incorporate the known factors, such as the size of the project and known future requirements (Serrador et al. 2015).

Agile Components

In the traditional waterfall methodology, each phase of the project is distinctly separate. The System Development Life Cycle, i.e. requirements, design, development, testing, and deployment, requires that each phase is completed individually before moving on to the next phase. The software must pass a number of quality checks after completing a phase and before moving on to the next phase. Each phase of the waterfall method has a specific deliverable, and the project has a predetermined go-live date. However, a major disadvantage is the difficulty returning to a previous phase and making significant changes. For instance, if a new requirement is found during testing, the requirement could impact the earlier design phase deliverables, causing changes that result in project delivery delays. Additionally, software products are not delivered incrementally, and the final product is implemented as a single release. Waterfall methodologies are well suited for predictable environments, i.e. heavily regulated, but are cumbersome, bureaucratic, and lack the capability to succeed in environments comprised of high uncertainty and change (Beck 1999). Traditional measurements of project success focused on the time, budget, and product quality (Atikinson 1999).

The Agile approach to software development originated to overcome the disadvantages of the waterfall methodology, primarily to shorten the development time for software and get the product to the customer / market as quickly as possible. Work is broken down into small, iterative cycles, known as sprints, with code releases built into the iterations. Each increment builds upon the previous until the total software product is complete. Agile is comprised of repeated practices that enable teams to work in a faster paced environment, such as behavior driven development, test automation, continuous integration, and continuous deployment. Each code release contains small portions of development necessary to build the larger framework. The measurement of a successful project is the overall end product delivered the customer and the subjective variable of the customer's satisfaction of the product's quality (Jugdev et al. 2005).

According to Serrador and Pinto (2015), projects utilizing the Agile methodology's iterative approach, report consistently higher project success. However, Agile software development emphasizes that teams should be self-managed and without direction on the implementation of leadership (Moe et al. 2009). A further challenge is the lack of synchronization of interdependencies among multiple Agile teams working on the same overall project (Melo et al. 2013). Teams with multiple autonomous members tend to find difficulty with the principle of the Agile manifesto, "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation" (Beck 2001).

Planning Ceremonies

Agile claims vast reductions in delivery based on the ability to adjust to changing business demands by operating in iteration cycles. However, the upfront planning is crucial to the team's success and is a process that cannot be eliminated from the development cycle. Planning ceremonies are used to determine the software release schedule and identify the chunks of work that fill the team's backlog. Each backlog item must be fully refined for the team's consumption and execution. The Agile terminology labels traditional project requirements as stories, which are groomed by working with the business stakeholder representing the team to fully convey the expected behaviors of the system. The depth of the groomed stories can be a determining factor in the success of the software code and can, also, contribute to the number of defects found in pre-production. Rigorous empirical analysis of the impact of story grooming is a key measurement of the effectiveness of agile software development.

Quality Impacts

From a practical perspective, using Test-Driven Development (TDD) and Behavior-Driven Development (BDD) approaches require that testing is performed simultaneously with development. TDD necessitates the creation of automated test scripts while developers are building the product. The test scripts become the requirements documents. BDD operates similarly to TDD; however, the language of the test scripts are basic programmable statements that can be automated using special software. BDD was designed to eliminate the complexity of building automated testing scripts. Specific phases of testing, such as full integration and user

acceptance testing, are not formally used. The product is accepted by the product owner, a business stakeholder, who typically previews a demo of the final version of the product before the code is released to production.

Along with the iterative development and testing, eXtreme Programming (XP) is frequently used in software development projects. The goal of XP is to accomplish significantly faster development through the collaboration of developers programming the same functionality simultaneously. During XP, stakeholders are present to review the product and provide immediate feedback. Favorable results have been reported in studies using XP as part of the Agile methodology, although few case studies have been performed providing documented, empirical evidence (Layman et al. 2006).

3. WATERFALL STRUCTURAL COMPONENTS

In contrast, waterfall projects follow several phases of testing to include a minimum of the following: unit/system, integration, regression, and user acceptance testing. Testing is a formal part of the project life cycle with documentation and traceability to the business requirements. While Agile appears to have a gap in the lack of full integration and traditional user acceptance testing, the prevailing industry argument maintains that smaller increments require less integration and software can be validated through repeatable automation tests. The Agile version of acceptance testing is achieved through the business' acceptance of the product at the end of the sprint demo.

Formalized Business Requirements

Testing is a validation that the system works according to the business requirements. Waterfall business requirements documents created traceability between test cases and specific requirements. Rather than producing business requirements documents, Agile creates stories that are recorded and serve as the business requirements. Many Agile tools, such as VersionOne, JIRA, GitHub, etc. provide linkages from test scripts to stories, thus solving for the traceability of requirements to testing. Due to the rapid timeline, acceptance tests are often foregone as too timely and requiring business resources that may not be readily available. When acceptance tests are performed, they are process driven and tend to be end-to-end. External systems are typically required to be fully functional and require considerable work to set up the environment properly before test execution (Rogers 2004).

Quality of software code is primarily measured by evaluating the number and criticality of production defects. To produce high software quality and reduce the number of defects, testing is seen as the solution for employing higher levels of code quality. Thus, the focus in Agile has shifted from scrum principles to ensuring test-driven development (TDD) and behavior-driven development (BDD) practices. The use of the Agile methodology affects perceived software quality through its impacts on internal performance (Kong 2007). When complemented with pair programming, an eXtreme Programming method wherein developers work side by side, teams are more productive and produce fewer defects (Rico et al. 2009). Test-driven development is posited to be "100 times more efficient than traditional methods when combined with continuous integration" (Rico et al. 2009). Rico provides return on investment (ROI) values to validate the claim, but does so without an empirical side by side comparison of the number of defects to software development hours.

4. CASE STUDY SCOPE AND RESEARCH OBJECTIVES

Research performed in previous studies of Agile has found that performance is linked with the effectiveness of teamwork coordination in software development teams (Moe 2009). The quality of the software code is of critical importance to organizations seeking to maintain a competitive advantage in the marketplace. This research seeks to provide an analysis of the defect rate of the Agile methodology and compare it to the defect rate using the Agile methodology across eight separate projects.

DigiTek LLC is a consulting firm that works with public and private sector companies. Data used for this study was gathered during consulting activities for two different clients, an insurance company and financial institution, and a total of eight project teams. This resulted in eight separate project teams from which to evaluate. The analysis was performed using three Waterfall and five Agile project teams of similar size. Each of the teams worked within the same line of business, each with separate functionality being developed and implemented.

The following research questions were raised:

1. Is the Agile Methodology able to complete a comparable number of software development hours while achieving higher levels of software code quality when compared to similar efforts using the Waterfall Methodology?

2. How is the criticality of defects impacted by Agile versus Waterfall?

Research Method

To establish comparable measurements between the Waterfall and Agile projects, software development hours were recorded along with production and pre-production defects. Agile projects operated in two-week sprint cycles for a total of twelve weeks, and Waterfall projects operated for a total of twelve weeks. Included in the study were two Agile project teams and two Waterfall project teams.

The following assumptions were made for the Agile teams participating in the study: Extreme Programming (XP) was used; stories were created and properly groomed; level of skill sets of developers, testers, and supporting team members were not a determinant in the study; and equal skill set was assumed across all participating teams.

Limitations to the study were recognized, as each of the teams worked on different software platforms and functionality. Due to the nature of rapid delivery methodologies, pre-production defects are not normally recorded. Agile teams work closely together to develop, test, develop, and test in eXtreme Programming sessions, fixing defects immediately. The ability to fix the defects and gain stakeholder approval real-time propels the team for faster turnaround of the final product. As a result, Agile teams have reduced reliance on pre-production metrics for determining quality. The focus becomes on the successful completion of iterative product releases and production code quality. For the purposes of this study, pre-production defects were recorded and used for the quantitative analysis. However, criticality was not recorded for pre-production defects regardless of methodology.

Case Study Procedure

Results from both a large financial institution as well as an international insurance company operating with Agile software development teams as well as Waterfall teams were used to evaluate the choice of project methodology's impact on code quality. Quality measurements were based on the number and criticality of production defects relative to the number of software development hours. Level of Agile maturity was measured for each team based on the following: grooming ceremonies were performed, software was released at the end of each iteration/sprint, each team had an assigned product owner, each team had an assigned scrum master, and daily scrums were held.

Participants included eight project teams, three of which utilized Waterfall and five that utilized Agile. Software development hours and defects were measured across the teams for a twelve-week period. With differing numbers of developers per project team, the ratio of software development hours to code quality allowed for an equivalent measurement. By employing a direct comparison of Agile and Waterfall projects, the analysis avoids the subjectivity and relativity of industry defect averages. Each of the project teams tracked their pre-production and production defects for code release during the twelve-week period. Additional measurements were performed to determine the impact of the level of agile maturity on the number of production defects. Product owners provided a level of acceptance of the product prior to each release to production by giving a go or no-go decision to release.

5. RESULTS AND ANALYSIS

Raw Defect Analysis

Pre-production defects, which were recorded for all teams during the twelve-week cycle, were much higher for Waterfall teams than for Agile teams, 43 to 13 respectively, with double the percentage of defects per software development hours, 1.37% to 0.63%, as shown in Table 2. This indicates a higher level of testing prior to implementation in order to prevent production defects. The amount of testing automation utilization was not considered in the quantitative analysis.

Defects by Project Methodology					
Project Methodology	Total Combined Development Hours	Total Number Prod Defects	Total Number Pre-Prod Defects	Percentage of Prod Defects to Dev Hours	Percentage of Pre-Prod Defects to Dev Hours
Waterfall	3140	19	43	0.61%	1.37%
Agile	2076	21	13	1.01%	0.63%
Full project cycle of 12 weeks					

Table 2
Defect analysis for Waterfall and Agile projects based on software development hours

Likewise, production defects were recorded for all teams during the twelve-week cycle. The total number of production defects for Waterfall compared to Agile were relatively the same, 19 to 21. However, when factoring in the number of software development hours for each project methodology, the percentage of production defects relative to software development hours for the Waterfall teams was nearly half the percentage of defects found in the projects using Agile. Waterfall projects had 0.61% of product

defects to development hours whereas Agile suffered 1.01%. The analysis indicates that the Waterfall methodology yields higher software development quality due to the time spent testing in pre-production regions.

Defects by Criticality

Total production defects recorded by all teams were further broken down by criticality. The criticality of defects was recorded as high, medium, or low, and criticalities were determined by the project managers. A general definition of high criticality represented defects that inhibited major functionality or contained significant user experience impacts. Medium criticality is assigned to defects with impacts to core functionality where alternative solutions exist. Low criticality defects do not prevent users from performing any intended functionality, such as font sizes on graphical user interfaces.

When reviewing the criticality by project methodology, fewer Waterfall defects were considered *high* criticality (Figure 1 and Figure 2). The results for Agile had the most difference when considering the *high* criticality defects, which made up 33% of their production defects compared to 26% for Waterfall. The *high* plus *medium* combined group signify defects that must be fixed prior to going to production. In considering the *high* plus *medium* criticality, the Agile projects yielded the same percentage of defects with 52%.

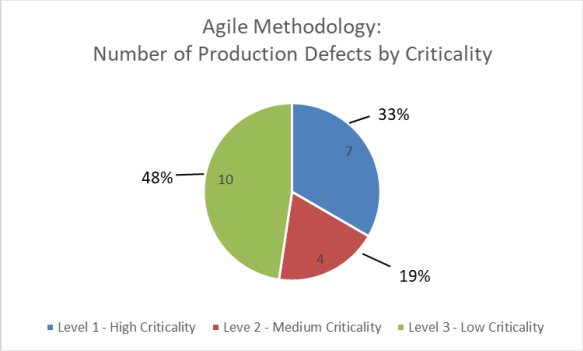

Figure 1
Defect analysis by criticality for Agile Methodology

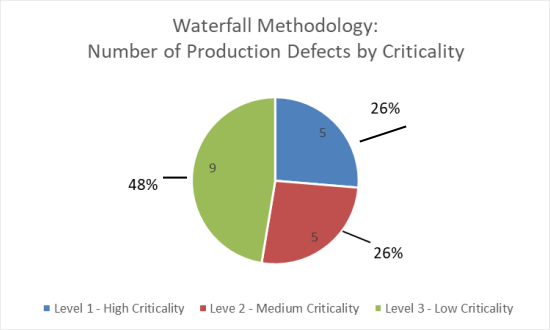

Figure 2
Defect analysis by criticality for Waterfall Methodology

The analysis between the two methodologies indicates that projects using the Waterfall methodology are more likely to spend more time testing, resulting in higher numbers of pre-production defects found and remediated prior to implementation. Agile's rapid delivery methodology finds fewer defects in pre-production but has double the production defects as a percentage of software development hours after implementation. The cause of the disparity can be related to numerous factors, such as the longer period of time spent testing in Waterfall projects contrasted with the targeted test cases performed during the Agile sprint cycle.

Agile project teams rely more heavily on automated test cases and spend less time on user acceptance testing. User acceptance testing, as well as manual testing, can allow the user/tester to focus efforts on attempts to break the application. Automated test scripts tend to focus on the simple, 'happy path', test cases rather than creating test scenarios for multiple permutations of the functionality. Automated test cases can be more accurate than manually running a test, because they are less subject to error. However, automated test cases sacrifice the depth of the testing.

6. CONCLUSIONS AND RECOMMENDATIONS

The overall results of this study suggest that Waterfall projects spend more time during the testing cycle and identify more defects prior to production. Agile projects have smaller pieces of functionality being delivered and focus on the speed of delivery, thereby shortening the volume of testing. Practitioners making selections between the two methodologies should consider the depth of pre-production testing. The Agile methodology can increase the software development quality through more expansive testing measures prior to releasing the software

code to production. The use of automation simplifies and shortens the testing cycle but should not be relied upon solely for testing all permutations of the changed functionality.

Future studies of automation could determine the impact of automated testing and behavior driven development on the code quality measurement. Additionally, this study did not review customer satisfaction of the products delivered but merely the defects.

7. REFERENCES

Atikinson, R. (1999). Project management: cost, time, and quality, two best guesses and a phenomenon, it's time to accept other success criteria. International Journal of Project Management, 17, 337-342.

Beck, K. (1999). Extreme Programming Explained. Boston: Addison-Wesley.

Beck, K. (2001, February 11). Principles Behind the Agile Manifesto. Retrieved October 25, 2017, from Manifesto for Agile Software Development: http://agilemanifesto.org/principles.html

Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: a systematic review. Information Software Technology, 50(9), 833-859.

Holbeche, L. (2015). The Agile Organization: How to Build an Innovative, Sustainable and Resilient Business. London: Kogan Page Limited.

Jugdev, K., & Muller, R. (2005). A retrospective look at our evolving understanding of project success. Project Management Journal, 36(4), 19-31.

Kong, S. (2007). Agile Software Development Methodology: Effects on Perceived Software Quality and the Cultural Context for Organizational Adoption. Ann Arbor: ProQuest Information and Learning Company.

Layman, L., Williams, L., & Cunningham, L. (2006). Motivations and measurements in an agile case study. Journal of Systems Architecture, 52, 654-667.

Melo, C. d., Cruzes, D. S., Kon, F., & Conradi, R. (2013). Interpretative case studies on agile team productivity and management. Information and Software Technology, 55, 412-427.

Moe, N. B., Dingsoyr, T., & Dyba, T. (2009, November 20). A teamwork model for understanding an agile team: A case study of a Scrum project. Information and Software Technology, 52, pp. 480-491.

Rico, D. F., & Sayani, H. H. (2009). The Business Value of Agile Softare Methods: Maximizing ROI with Just-in-Time Process and Documentation. Fort Lauderdale: J. Ross Publishing.

Rogers, R. O. (2004). Acceptance Testing vs. Unit Testing: A developer's perspective. LNCS (3134), 22-31.

Sampaio, A., Vasconcelos, A., & Sampaio, P. F. (2004, November). Assessing Agile Methods: An empirical study. Journal of the Brazilian Computer Society, 10(3).

Serrador, P., & Pinto, J. K. (2015). Does Agile work? - A quantitative analysis of agile project success. International Journal of Project Management, 33, 1040-1051.

Appendix

	Waterfall methodology	Agile methodology
Core delivery process	Follows the system development life cycle with requirements analysis, design, development, testing, implementation, and support pre-defined cycles with a fixed end date	Iterative requirements, design, and development with continuous integration and deployment activities and iterative release cycles
Stakeholder Involvement	Fixed project scope with stakeholder approver before design and development can begin; stakeholder review of final product	Changing scope per sprint iteration with stakeholder input on requirements and prioritization; stakeholder review of final product
Development	Based on fixed requirements and overall architectural design	Performed while determining requirements and test-driven
Testing	Largely manual execution, tied directly to business requirements, performed as several phases (unit, integration, user acceptance, performance)	Automated, behavior-driven, performed as part of development
Documentation	Business and system requirements are written and approved by stakeholders; testing scenarios and results are documented with traceability to requirements	Requirements are loosely documented as sprint stories; testing scripts are sometimes tied to stories

Table 1
Methodological distinctions between Waterfall and Agile project methodologies