This article is part of the IEEE Reliability Society 2008 Annual Technology Report.

Customer Satisfaction and Agile Methods

Donald L. Buresh
Email: LoganSquareDon@sbcglobal.net

A Review of the Results of the Study by Donald L. Buresh

Introduction

From February 11th to the 13th of 2001, at The Lodge at the Snowboard ski resort in the Wasatch Mountains of Utah, 17 men, no women, met to relax, ski, and talk about software development. The group named itself the "Agile Alliance", and after two days of meetings and some fun, these 17 people were surprised to find how much they had in common [1]. They agreed that the core of their software philosophy was a set of values based on trust and respect, where good products are delivered to customers by individuals who are the most important assets of the companies employing them [1]. The output of the meeting was entitled a *Manifesto for Agile Software Development*, and was signed by all of the participants. There was one concern voiced by Martin Fowler, a British subject, who felt that most Americans would incorrectly pronounce the word "agile" [1].

Literature Review

Agile Manifesto

The *Agile Manifesto* was essentially a statement of values and principles that bears to be repeated. It said:

> We are uncovering better ways of developing software by doing it and helping others doing it. Through this work we have come to value: Individuals and interactions over process and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, Responding to change over following a plan. That is, where there is value in the items on the right, we value the items on the left more. [2]

The *Agile Manifesto* continued by stating a set of principles that formed the basis of the agile-driven perspective. The principles are the pillars that support this revolutionary document along with the resulting movement that is currently dominating the software development culture. The first and foremost principle of the agile movement is that customer satisfaction is of the highest priority, and is achieved via the rapid and continuous delivery of useful and valuable software [2].

Agile-driven methods are a response to the fact that most software development projects are managed poorly, resulting in significant cost overruns and serious delays in the delivery of the final product [3]. One of the more fruitful process-oriented methodologies is Goldratt's [4] Theory of Constraints, where the challenge was to translate Goldratt's techniques to software engineering and other computer-related fields [5]. The issue is that many of the salient features of the theory could be lost when superimposing it on a software development process [6]. Boehm and Turner believed that software engineering practices are at times cryptic and do not readily lend themselves to the imposition of practices from other industries.

It is not that software development and software engineering are uncontrollable processes; but rather, the purported power of agile-driven methods is that it permits a project manager to focus on responding to and achieving value quickly [6]. The value of agile-driven software development methods is in the understanding of what variables can be tracked, and what

variables can be safely ignored, so that appropriate and timely information is available to customers and senior management.

The *Agile Manifesto* is a simple declaration consisting of 12 principles [2]. The principles were designed to generate software in a timely manner, building a project around motivated individuals, conveying information effectively, maximizing the amount of work done, and most importantly, satisfying the customer. Although the goals of the manifesto seem to be commendable, it may be that only marginal results can be obtained in practice because the field is in a state of apparent flux [6]. According to Anderson [5], agile-driven methods for software development projects are essentially a methodology for applying the Theory of Constraints to achieve the desired business results. Thus, it seems that the Theory of Constraints may be directly applicable to software development projects, where the focus is on maximizing throughput, while minimizing operating expenses and inventory [4],[5]. The translation of these process-oriented concepts to software development is however fraught with some risk, because software developers may find the imposition of the methodology to be unwarranted [5].

Characteristics of Agile-Driven Methods

Boehm and Turner [6] classified agile-driven methods according to its characteristics, and then proceeded to provide concrete examples of these different techniques. Agile-driven methods were defined as "lightweight processes that employ short iterative cycles; actively involve users to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation" [6 (p. 17)]. The implication is that for a process to be truly agile, it must be iterative over several cycles, incrementally delivering the product. It must employ self-organizing teams that manage themselves by determining the best way to do the work. Finally, the processes, principles, and work structures must emerge during the course of the project, rather than be pre-determined [6].

Agile-driven methods can also be viewed from an operational perspective. Agility embraces change, making it an ally and a friend rather than a hard and cruel taskmaster. Agile-driven methods promote creativity within an agile-driven software development team by delivering exceptionally fast value to the customer [6]. In an agile-driven project, many releases of a software application are scheduled. The intent is to ensure that only the highest priority functions are implemented first, thereby quickly delivering value to the customer, and promoting the timely emerging of requirements [6]. The key to employing agile-driven methods is to strip a design to what is just currently being developed with the hidden agenda that change is inevitable, and that planning for future functionality is a waste of time and energy [6]. Another feature of agility is that a software product may need to be restructured to remove duplication of code, to improve communication among modules and/or subsystems, and to add flexibility without changing the behavior of the application [6].

According to Boehm and Turner [6], when generating the source code of a software application, agile-driven methods can employ paired programming teams, where two programmers worked side by side, collaborating on the same design, algorithms, code, or test procedures. In an empirical study conducted at the University of Utah, Erdogmus [7] observed that there was a significant economic advantage to paired programmers over solo programmers. However, this is not necessarily a surprising result, because increasing returns to scale is a well-known microeconomic phenomenon that occurs when companies or teams are very small.

Other features of agility include an evaluation of the work performed, the methods employed, and estimates made after an iteration is completed [6]. The purpose of the action is to encourage teams to learn and reflect upon what happened in the past so that estimates of future activities are

more accurate. Thus, tacit knowledge is gained by the project team members rather than being lost or having the information being recorded in a document [6]. When it would be imperative that written documentation exist, such as in a classified or highly sensitive project, this feature of agility could possess some serious negative implications [6].

Characteristics of Plan-Driven Methods

According to Boehm and Turner [6], plan-driven methods are generally thought to be traditional ways to develop software. The methods approach software development from the perspective of mainline engineering fields, where the requirements/design/build paradigm promotes orderly processes focus on continuous improvement [6]. The general idea behind a plan-driven approach is based on the tenets of systems engineering, where a large number of components are produced by a single company employing a cadre of workers. The advantage of a plan-driven strategy is that it works well when developing hardware components, but software development may be a horse of a different color. Software has logical constraints rather than physical constraints [6]. The result of this dichotomy is that software projects are typically late, over budget, and of poor to mediocre product quality. In response, the Department of Defense developed a series of documents to help software developers create applications for the defense establishment, including MIL-STD-1521, DoD-STD-2167, and MIL-STD- 498 and their successors [6]. In the private sector, companies such as IBM, Hitachi, and Siemens also developed similar standards [6].

Because plan-driven methods are characterized by a systematic approach to software development, the software application usually moves through a series of well-defined steps from requirements to finished executable code. Thus, it becomes paramount that the documentation thoroughly authenticates the workings of a software application, regardless of whether the development cycle follows a waterfall model or is rather incremental or evolutionary in nature [3]. One of the problems with plan-driven methods is that software developers view documentation as an anathema, sometimes openly rebelling against the need for meticulous documentation, and at other times, carelessly and insidiously ignoring the necessity for accurate and precise records.

Because software is malleable, plan-driven software development methods scrupulously define, standardize, and incrementally improve software processes, detailed plans, activities, workflows, roles and responsibilities, and work product descriptions [3]. The implication is that individuals are trained to monitor, control, and educate the infrastructure [3]. The advantages of plan-driven methods are derived from the Industrial Revolution, where specific processes were repeated, specific work performed, and where any individual could examine and understand the work accomplished. Thus plan-driven software development methods allow individuals to move from project to project without a significant amount of retraining, where the loss of a key contributor does not adversely affect the success of a project.

One of the key concepts of plan-driven software development methods includes *process improvement*, where a group of activities are designed to improve the performance and maturity of an organization's processes [3]. Another key notion is that of *process capability* or the "inherent ability of a process to produce planned results" [3 (p. 12)]. In other words, as a process improves, it becomes more measurable and predictable so that product quality is enhanced and productivity is assured. Furthermore, as would be expected, organizations mature in the sense that processes become standardized across the great divide that sometimes separates functional departments, ensuring that common assets are effectively deployed. Specialists congregate together to ensure the integrity of the processes employed, and to measure and assess the uncertainties that can result in a significant loss or harm to the firm [3]. *Verification* confirms that specifications, designs, and models reflect the specified requirements, whereas *validation* is employed to determine the fitness of a product to its mission [3]. Finally, to achieve these

characteristics, a software system architecture is created to identify the software and system components, connectors, and constraints; to understand the needs of the system stakeholders; and to show how the system components, connectors, and constrains actually satisfy the needs of the system stakeholders [3].

It should not be forgotten that strict planning can impede innovation, and thus become the object of stifling asphyxiation. The danger of plan-driven software development methods is that its team members can become like mechanical androids being so focused on the process that he or she loses sight of the big picture [3]. On the other hand, plan-driven methods that have managerial support, organizational infrastructure, and seasoned practitioners can thwart the onslaught of an unnecessary concentration of effort. Some of the advantages of plan-driven software development methods include the existence of component libraries that encourage software reuse, and a well-trained staff that understand the value of creating and maintaining effective documentation [3].

Research

The research conducted by Buresh [8] was used to assess whether the use and results agile-driven software development methodologies are as effective in satisfying customers as the use and results of plan-driven software development methodologies. Christensen [9] suggested that the reason projects fail is because the customary answers of planning better, working harder, and becoming more customer driven actually exacerbate the problem. The whole purpose of plan-driven project management techniques is to capture and satisfy customer needs at the beginning of a project. Thus, agile-driven methodologies may be better able to address the needs of the customers throughout a project, thereby ensuring its continued success.

One of the major goals of agile-driven methods is the incorporation of customer input to ensure timely delivery of software products [2]. The focus is on satisfying the customer throughout the software development cycle. In contrast, in plan-driven software development methodologies, the customer reviews the progress of the software being developed either at pre-established milestones or at the end of the project [10]. The difference in customer involvement is critical, and leads directly to the statement of the problem that was examined in this study.

Research Question

The research question guiding Buresh [8] was: What is the difference, if any, in customer satisfaction between the use and results of agile-driven software development methods and the use and results of plan-driven software development software development methods? The data in Buresh [8] were analyzed to explore the following hypotheses:

> H0: The autonomous customer satisfaction for a project using agile-driven software development methods is equal to the autonomous customer satisfaction for a project using plan-driven software development methods.
> $H1_a$: The autonomous customer satisfaction for a project using agile-driven software development methods is greater than the autonomous customer satisfaction for a project using plan-driven software development methods.
> $H1_b$: The autonomous customer satisfaction for a project using plan-driven software development methods is greater than the autonomous customer satisfaction for a project using agile-driven software development methods.

The statistical procedure examined if H0 and $H1_b$ could be rejected at the 95% or 99% confidence levels. If so, it meant that autonomous customer satisfaction for projects using agile-driven software development methods was significantly greater than autonomous customer satisfaction for projects using plan-driven software development methods. The research regressed customer

satisfaction, the dependent variable, against product quality, project team effectiveness, and project management effectiveness, the three independent variables. A dummy variable was added to the regression equation, where the dummy variable equaled zero for plan-driven software development projects, and equaled one for agile-driven software development projects.

The four variables employed in Buresh [8] include customer satisfaction, the dependent variable, and product quality, project team effectiveness, and project management effectiveness, the three independent variables.

Customer Satisfaction

Varva [11] viewed customer satisfaction as either an *outcome* or a *process*. The outcome portion of customer satisfaction dealt with the customer being adequately or inadequately rewarded [12]. In other words, it was an emotional response to the experiences provided by the customer's association with a software development project (Westbrook & Reilly, 1983). Customer satisfaction was also the outcome of paying for a software development project, whereby the customer compares the results of a project with the expected consequences (Churchill & Surprenant, 1982).

In order to understand the degree of customer satisfaction that exists, it is important to determine the customer requirements for a project using agile-driven or plan-driven software development methods. One way was to collect a series of critical incidents. According to Hayes [13], a critical incident is a specific example of how software development methods performed in a positive or negative manner. A good critical incident is not only specific, but also describes software development methodology in behavioral terms, using specific adjectives [13]. The idea is that a critical incident describes a single behavior that can be interpreted in the same way by different people.

When examining customer satisfaction for agile-driven and plan-driven software development methodologies, it was important to ask not only whether the customer was satisfied with the results of a software development project, but also if the customer would use the same software development methodology in the future. A second criterion was whether a customer would recommend the software development methodology to a friend, or a business associate. Another criterion was the worth of a software development methodology in terms of quality, cost, and its ability to contribute to the completion of a project in a timely manner compared with the ability of other projects similar in size.

Product Quality

For Hayes [13], there were two approaches to finding critical incidents - group and individual interviewing. After obtaining the list of critical incidents, the results from the two approaches would be analyzed to find common ground. Next, the customer requirements that were obtained would be used to define the quality of the product. According to Hayes [13], correctness, reliability, usability, maintainability, testability, portability, inter-operability, intra-operability, and flexibility were the relevant software quality issues. Correctness was the degree to which the software product satisfies customer specifications. Reliability was the extent that software product performed its intended functions [13]. Usability was defined as the effort needed to understand the product resulting from a software development project. Maintainability was the effort required to correct mistakes or errors, whereas testability was the amount of work required to ensure that a software product performs its intended functions [13]. Portability was the process of transferring the software development methods from one project to another project. Inter-operability was the ability to coordinate multiple projects using similar or different software development methods, and intra-operability was the capacity for the components of a software

product to communicate with each other [13]. Finally, flexibility was the capability of using a given software development methodology in other projects.

Project Team Effectiveness

Project team effectiveness deals with the human element of a project. It includes the responsiveness of the project team to customer issues, the speed in which customer issues are incorporated into the software product, the availability of the team to focus on customer requirements and issues, the professionalism of the project team, and the enthusiasm of the project team [14]. These issues were examined in this study because a major assumption of agile-driven methods is that customers actively interact with the project team throughout the course of a project. Customers are not merely bystanders who cheer the project team onward to success, but are considered members of a project team, guiding the team to a successful conclusion [3].

Project Management Effectiveness

The last independent variable is project management effectiveness. It is a standard issue in any project, regardless of the methodology employed. According to Kerzner [14], project management effectiveness can be expressed in terms of the following questions:

1. Did the project finish in the stated timeframe?
2. Was the project planning complete from beginning to end?
3. Did the project manager understand how much time the project required?
4. Was the project completed under, on, or above budget?

In Buresh [8], the answers to the above questions were weighed against similar responses from projects that did not use agile-driven methods. If the answers had not been statistically significant when compared with other software development methods, it implied that customers were satisfied with the agile-driven methodology employed, but would have been equally satisfied with using plan-driven methods. Buresh [8] verified that not only was the customer satisfied with the use and results of agile-driven software development methods, but also that the customer was satisfied over and above his or her satisfaction level had an organization used a plan-driven software development methodology.

Results

In Buresh [8], it was examined empirically whether the use and results of agile-driven software development methods significantly satisfied the customer over and above the use and results of plan-driven software development methods. Customer satisfaction was regressed against product quality, project team effectiveness, and project management effectiveness, where a dummy variable was used to distinguish between agile-driven projects and plan-driven projects. All of the variables contained averaged values. One hundred eighty-five national and international participants reported on 95 known agile-driven projects, 53 known plan-driven projects, 24 probable agile-driven projects, and 13 probable plan-driven projects. When the software development methodology was known by the participants, there was no statistically significant difference in autonomous customer satisfaction between the use and results of agile-driven software development methods and the use and results of plan-driven software development methods at the 95% confidence level. All of the non-standardized coefficients of the explanatory variables were statistically significant at the 99% confidence level. When the software development methodology was known by the participants or inferred by the study, the non-standardized coefficient of the dummy variable was barely statistically significant at the 95% confidence level, but not statistically significant at the 99% confidence level. The non-standardized coefficients of the explanatory variables were again statistically significant at the 99% confidence level.

Because customer satisfaction, product quality, project team effectiveness, and project management effectiveness contained averaged values and because the [15] correction procedure was invoked, Buresh [8] argued that the 99% confidence level was the appropriate significance level to employ. Thus, when the software development methodology was known by the participants or inferred by the study, there was again no statistically significant difference in autonomous customer satisfaction between the use and results of agile-driven software development methods and the use and results of plan-driven software development methods.

Conclusions

The significance of the study conducted by Buresh [8] stemmed from the fact that little empirical research has been conducted in establishing whether customer satisfaction in the use and results of agile-driven software development methods was greater than the customer satisfaction in the use and results of plan-driven software development methods. Most of the literature has been either anecdotal in nature, case studies, or consisted of a simulation [16]. Furthermore, there has been no research verifying whether the customer satisfaction in the use and results of plan-driven software development methods is greater than the customer satisfaction in the use and results of agile-driven software development methods. Because it turned out that the use and results of neither software development method yielded customer satisfaction levels greater than the other, the outcome is significant. It indicated that both methods satisfy their respective customers under a wide range of different situations. The research by Buresh [8] is significant because it helps delineate when and where to employ agile-driven and plan-driven software development methodologies.

Finally, Cosby [17], Sieber [18], and Trochim [19] have all observed that case studies and other anecdotal discussions have no external validity. In other words, the extrapolation to other instances should not be made when the basis of the extrapolation is a case study. Buresh [8] brings this point home to roost by showing that the data collected does not support the proposition that the use and results of agile-driven software development methods provide greater customer satisfaction than the use and results of plan-driven software development methods. In some sense, Buresh [8] confirms the methodological concerns voiced by Cosby , Sieber, and Trochim [17]-[19]. It remains the responsibility of the signers of the *Agile Manifesto* and the agile community to verify the propositions of the manifesto regarding customer satisfaction by using statistical analysis instead of case studies.

References

[1] Highsmith, J. (2001). History: The agile manifesto. Retrieved on September 23, 2006 fromhttp://www.agilemanifesto.org/history.html

[2] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Manifesto for agile software development.* Retrieved April 7, 2004, from http://agilemanifesto.org/

[3] Highsmith, J. (2004). Agile project management: Creating innovative products. Reading, MA: Addison Wesley Longman.

[4] Goldratt, E. (1990). What is this thing called theory of constraints and how should it be implemented? Great Barrington, MA: North River Press.

[5] Anderson, D. J. (2004). *Agile management for software engineering: Applying the theory of constraints for business results*. Upper Saddle River, NJ: Pearson Education.

[6] Boehm, B. & Turner, R. (2004). Balancing agility and discipline: A guide for the perplexed. Boston: Pearson Education.

[7] Erdogmus, H. (2003). The economics of software development by pair programmers. The Engineering Economist, 48(4), 283-319.

[8] Buresh, D. (2008). Customer satisfaction and agile methods: Assessing customer satisfaction and agile project management methods. Germany: VDM Publishing House Ltd.
[9] Christensen, C. M. (1997). The innovator's dilemma: The revolutionary book that will change the way you do business. New York: Harper Collins Publishers.
[10] Desaulniers, D. H., & Anderson, R. J. (2001). Matching software development life cycles to the project environment. Proceedings of the Project Management Institute 32nd Symposium, 2001, Nashville, TN, CID: 591.
[11] Varva, T. G. (1997). Improving your measurement of customer satisfaction: A guide to creating, conducting, analyzing, and reporting customer satisfaction measurement programs. Milwaukee, WI: American Society for Quality.
[12] Howard, J., & Sheth, J. (1969). The theory of buyer behavior. New York: John Wiley & Sons.
[13] Hayes, B. E. (1998). Measuring customer satisfaction: Survey design, use, and statistical analysis methods (2nd ed.). Milwaukee, WI: American Society for Quality.
[14] Kerzner, H. (2006). Project management: A systems approach to planning, scheduling and controlling (9th ed.). New York: John Wiley & Sons.
[15a] Bonferroni, C. E. (1936). Teoria statistica delle classi e calcolo delle probabilità. Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze, 8, 3-62.
[15b] Bonferroni, C. E. (1935). Il calcolo delle assicurazioni su gruppi di teste. In Studi in Onore del Professore Salvatore Ortu Carboni. Rome, Italy, 13-60.
[16a] Cao, L. (2005). Modeling dynamics in agile software development. ProQuest Information and Learning Company. (UMI No. 3197587)
[16b] Cao, D. B. (2006). An empirical investigation of critical success factors in agile software development projects. ProQuest Information and Learning Company. (UMI No. 3238566)
[17] Cosby, P. C. (2004). Methods in behavioral research (8th ed.). Boston: McGraw-Hill.
[18] Sieber, J. E. (1992). Planning ethically responsible research: A guide for students and internal review boards. Newbury Park, CA: Sage Publications.
[19] Trochim, W. K. (2001). The Research methods knowledge base (2nd. ed.). Cincinnati, OH: Atomic Dog Publishing.
[20] Project Management Institute. (2005). A guide to the project management body of knowledge: PMBOK guide 2005 edition. Newton Square, PA: Project Management Institute.