2009

The Challenge of Improving Software Quality: Developers' Beliefs about the Contribution of Agile Practices

Sue Kong
Kutztown University of Pennsylvania, kongsue@gmail.com

Julie E. Kendall
Rutgers University, julie@thekendalls.org

Kenneth E. Kendall
Rutgers University, ken@thekendalls.org

The Challenge of Improving Software Quality: Developers' Beliefs about the Contribution of Agile Practices

Sue Kong
Kutztown University
kongsue@gmail.com

Julie E. Kendall
Rutgers University
julie@thekendalls.org

Kenneth E. Kendall
Rutgers University
ken@thekendalls.org

ABSTRACT

We observe that systems developers who have had experience with agile development projects often express their opinion that agile methodologies are superior to plan-driven methodologies. In order to collect empirical evidence to support or discount this belief, we conducted a survey among software developers about software quality and development practices. Our study identified eight quality goals from the software quality literature. We then asked the participants to identify which of eight practices contributes the most towards that quality goal. Half of the practices were agile practices; half were plan-driven practices. We found that, for each and every quality goal, the participants as a whole chose one of the agile practices as a best practice enabling them to reach each quality goal. While this study does not conclude that agile methods are always the best approach, it does reveal that agile practices are being noticed and appreciated by many system developers.

Keywords: Agile Software Development, Agile Practices, Software Quality, Software Design Quality, Software Coding Quality, Survey of Software Developers.

INTRODUCTION

Plan-driven methodologists, systems analysts and developers using plan-based methodologies such as the SDLC have argued that agile methodology leads to software quality degradation because its constant adaptation introduces errors and defects (Paulk, 2001, 2002). Examining software quality by two separate dimensions, that is, coding quality (coding conforms to software design) and design quality (software design confirms to customer requirements), we are able to provide a different point of view. Even though the constant adaptation of agile methodology does introduce coding defects, these defects can be eliminated effectively by intensive code review and frequent testing. On the other side, the iterative development cycles of the agile methodology highly encourage customer feedback, thus locating and eliminating design defects and ensuring better design quality.

Our analysis of agile methodology's strengths in improving software quality is supported by a variety of theoretical and survey studies. Some software practitioners have made claims that agile methodology has positive impacts on software quality. In addition, they argued that agile practices, such as pair-programming, test-driven programming, and continuous integration, have integrated quality control and assurance functionality and will improve software quality effectively (Armitage, 2004; Huo, Verner, Zhu, & Babar, 2004; Opperthauser, 2003). We also have found three surveys conducted recently that contain questions concerning the perceived impacts of agile methodology on software quality. We compare the details of these three surveys in Table 1.

Focus of the Survey	Target Population and Number of Responses	Related Findings
To determine how agile processes are being implemented; (Agile Alliance and VersionOne)	VersionOne's customer base and newsletter list; Agile Alliance members; readers of the *Agile Journal* and some other technology sites; N=722	74% of respondents reported enhanced software quality as one of their development values. 86% of respondents reported that software defects were reduced by more than 10%.

To investigate how many people are using agile development, what they are doing, and whether they are actually benefits from it;
(Scott Ambler)
	Mailing lists from Dr. Dobb's Journal and Software Development;
	N=4232
		66% of respondents reported higher quality when using agile development.
		58% of respondents reported improved satisfaction when using the agile approach.

To assess the state of agile adoption as well as to identify the drivers and barriers towards agile adoption.
(Digital Focus)
	Agile 2005 conference participants; Agile websites readers;
	N=136
		45% of IT professionals perceive agile development as the solution to scope management.
		47% of IT professionals perceive agile development as the solution to addressing unclear business requirements.
		17% of IT professionals perceived the agile methodology as the solution to improve code quality.

Table 1 Three Surveys on Agile Software Development (Agile Alliance and VerisionOne, 2006; Scott Ambler, 2006; Digital Focus, 2005.)

Analyzing and comparing the results of these studies, we found that these instruments did not differentiate design quality from coding quality. Additionally, they were lacking specifics on linking particular practices to achieving particular quality goals. Since software quality has multiple layers of meanings and aspects, we believe it will yield more insights and meaning if we examine software quality using different dimensions.

We therefore designed a questionnaire that measures the perceptions of effectiveness of different software practices in achieving various quality goals. The software practices we investigated include both agile practices and plan-based software practices. The software quality dimensions under investigation include coding quality, design quality, as well as customer satisfaction. We conducted a survey study among software developers to find out what agile practices are the most effective in improving different aspects of software quality. We provided a list of software practices, including both agile practices and plan-driven software practices, to survey respondents and ask them to identify what they perceived as the most effective practices in achieving various quality goals. We distributed 85 questionnaires and collected 57 usable responses. In an upcoming section we discuss the results, analysis, and findings of our survey.

AGILE PRACTICES

The most well known agile method is extreme programming (XP) proposed by Beck (1999). Other agile methods include Scrum (Schwaber & Beedle, 2001), Open Source Development (Sharma, Sugumaran, & Rajagopalan, 2002), Feature Driven Development (Coad, Lefebvre, & De Luca, 1999), Crystal family (Cockburn, 2002), and many other less well-known approaches. Many consider the plan-driven methodologies and the agile methodology to be opposites, suggesting that the former adds control and formality, while the latter causes liberation and revitalization.

The common features of these so-called "lightweight methods" include emphasis on adaptability when facing change or uncertainty, the utilization of competent people instead of rigorous processes to achieve flexibility and quality, the institution of fast delivery and less formality, and brief documentation. The commonalities inherent in these lightweight methods prompted their originators and practitioners to form an organization called the "Agile Alliance" to promote their common values, principles, and key practices. They also reached a consensus and generally categorized their methods as "Agile Software Development Methodology" (ASDM) as described in Beck, et al. (2001).

As such, agile software development methodology is a set of software development methods whose adherents share common values and principles. According to Beck et al. (2001), the agile methodology deals with environmental uncertainty and change by espousing the following values:

1. People and interaction, instead of processes and tools, are the most important factors for project success.

2. Working software, instead of documentation, should be the focus of the development effort and the only measure of progress.

3. Customer collaboration, instead of contract negotiation, is more effective for achieving project success.

4. Responding to change is more important than following a plan in turbulent environments.

These values reveal the theoretical divide between agile methodology and plan-driven methodologies. In contrast to plan-driven Systems Development Life Cycle (SDLC) or waterfall approaches, agile methodology, based on the belief that human error is the main reason for any project failure, puts people ahead of process. They argue that the most important contribution to the delivery of quality software within time and budget is the effective collaboration among individual stakeholders (Cockburn & Highsmith, 2001). In the agile software development methodology, the sequential importance of the software development processes is no longer emphasized. Agile software development methodology uses an iterative approach to support requirement changes and enable a project team to incorporate customer feedback into the design. Even if some agile software development methods do have process and a set of practices, they are often adaptable because the agile methodology is inherently self-adaptive.

Agile practices are a set of software practices guided by the agile values and principles. We use the XP practices that have been referred to as a typical set of agile practices as devised by Beck (1999. Agile practices underscore the importance of increased communication among people as well as the value of flexibility, quickness, and nimbleness. Some typical agile practices are listed in Table 2.

Which agile practices do systems developers believe are the most effective in improving software quality? In order to answer this question, we first examine what software quality means as well as how it is measured.

SOFTWARE QUALITY DEFINITIONS

Quality definitions originated from the manufacturing industry. Definitions of "quality" include "fitness for use" and "customer satisfaction" (Juran, 1992; Juran & Godfry, 1999; Gryna & Juran, 2001). Similarly, the Institute of Electrical and Electronics Engineers (1987) defines "software quality" as "the totality of features and characteristics of a software product that bear on its ability to satisfy given needs."

In the beginning of quality control and later quality assurance, especially in manufacturing, the standard was zero-defects. Note that this was a post-production measurement. In the modern software industry, software quality is no longer limited to zero-bugs. For example, Ray, a software project manager, whom we interviewed for an earlier case study, said:

> For application, by doing short iteration, it makes sure that it does what the customer wants. If you produce a zero defect product that does not do what the customer wants, then it is no good. There are multiple aspects to consider there. And I do not think agile will compromise quality if you do things correctly. In fact, it could be better because you are testing things more often.

Inspired by these comments, we noticed the following critical aspects of software quality: the design quality (systems design conforms to user requirements) and the coding quality (coding conforms to systems design). Traditional research on software quality has been highly focused on the coding quality, while ignoring the design quality. This way of equating non-bug software to high quality software has been so influential that many people still use the term software quality to refer to coding quality even now. It is important to separate these two aspects of software quality because they pose different challenges.

In addition, different types of software pose different requirements for coding and design quality. Highly critical software (software facilitating space missions or other life-threatening tasks where coding defect or design defects can cause significant losses) possesses a high standard for both coding and design quality; while innovative software with low criticality (software aiming to give the user a competitive edge by design; if it crashes, it can be restarted with little loss) has a high standard for design quality only.

Agile Practices	Definition
Onsite Customer	Have an actual user on the team full-time to answer questions. The customer, developers, and management work together to decide project scope, quality priority, and schedule, based on the business requirements and technical possibilities.
Short Releases	Deliver working version as often as possible. Incrementally add more features or improve quality according to feedback.
Minimal Documentation	Document only essential design and change. Most communication is done verbally in order to speed up the process.
Design Simplicity	Solve problems with minimal effort. Prioritize simple and easy to understand ideas.
Pair Programming	Carry out all development tasks by a pair of developers simultaneously. Often one developer comes up with ideas, while the other developer checks for defects and conformance to design guidelines.
Test Driven Programming	Intensively test the functionalities and design quality at the end of each iteration. Customers and testers need to provide valuable feedback to the development team in a timely manner so that the development team can incorporate the feedback into the objectives of the next iteration.
40-hour Week	Encourage members of the development team to work within their cognitive and physical limits. Do not overwork the team because people tend to work more, not better, under pressure.
Continuous Integration	Integrate the finished codes continuously into a compliable version of the systems to see if it fits. The integration may happen every day. Unit and acceptance tests are conducted at the integration to ensure that the current build of the systems is capable of passing all pre-specified tests in current form.
Collective Ownership	Allow everyone on the team to change the codes if needed.
Refactoring	Allow the team members to periodically work together to improve the systems design.

Table 2 Typical Agile Practices (based on Beck, 1999)

Once these two different quality aspects are separated, it is easy to see that the agile methodology has great strengths in detecting and eliminating design defects to ensure the systems design conforms to the user requirements. Onsite customer and short releases are considered very effective in capturing requirements and incorporating customer feedback into the design, leading to software of high value to customer. In addition, pair programming and continuous integration have been recognized as the best practices for eliminating coding errors. The plan-driven systems development methodologies lack similar mechanisms for detecting nonconformance to the user requirements, leading at times to software of high coding quality but low design quality.

In addition, software product quality is under the direct influence of process quality. From requirements gathering, systems design, systems development, to final release, the quality of each step is critical to the resultant software quality. For instance, if customer requirements are not captured properly, or the systems design does not address customer requirements, or the codes do not conform to the systems design, or changes in requirements are not captured and responded to, the resultant software will not possess good quality.

After reviewing software quality literature, we believe it is beneficial to measure the effectiveness of the agile practices in achieving the following software quality goals:

1. The effectiveness of agile practices in capturing customer requirements;

2. The effectiveness of agile practices in assuring that the systems design meets customer needs;

3. The effectiveness of agile practices in eliminating coding errors;

4. The effectiveness of agile practices in eliminating design defects;

5. The effectiveness of agile practices in improving responsiveness to customers' changing needs;

6. The effectiveness of agile practices in assuring that the systems design meets customer needs;

7. The effectiveness of agile practices in assuring that the resultant systems product provides values to end users;

8. The effectiveness of agile practices in satisfying customers.

As suggested in the work of Armitage (2004), Beck and Andres (2004), Huo et al. (2004) and Opperthauser (2003), agile practices such as the 40-hour week, customer involvement, pair programming, design simplicity, onsite customers, continuous integration, short releases, pair programming, collective ownership, and refactoring, all have integrated quality control functionalities to reduce coding errors and design defects. Hence, we included these practices in the questionnaire. Agile practices which we do not believe are directly related to quality management were not included. For example, no literature has suggested that minimal documentation leads to better quality performance. Therefore, minimal documentation practice was not considered in this study. Other omissions are developer productivity and fast delivery, which are not directly related to the quality effects.

We designed a questionnaire to capture all the measures we previously introduced and conducted a survey among software developers to find out what agile practices they perceive to be the most effective in achieving various software quality goals. Part II of the questionnaire collected demographic information, intended for cluster analysis.

CONDUCTING THE SURVEY

As suggested by Straub (1989), Moore and Benbasat (1991) and Chin, Gopal and Salisbury (1997), our survey study has followed the following three stages: content validation, pilot study, and full-scale survey.

1. Content Validation. After the pilot questionnaire was created, the items in the questionnaire were reviewed by all researchers, by five software professionals, and four academicians involved with managing innovation and software development to ascertain content validity through examining whether the items appropriately reflect the construct of interest. With the help of these experts, ambiguous items were identified and reworded.

2. Pilot Study. Before conducting the full-scale survey, a pilot study involving 10 software researchers and practitioners was conducted. The purpose of the pilot study was two-fold—to see how long it took respondents to complete the survey and to examine construct validity. From the pilot study, we learned that the survey took about 15-20 minutes to complete.

In addition, we found that most constructs showed good validity with the exception of several indicators that did not load well on their assigned construct. However, we decided not to delete these indicators at this stage because the sample size of the pilot study was 10, too small to yield reliable results. In other words, the phenomena we observed in the pilot study could be extreme cases; these indicators might load on their assigned construct well once the sample size increased. Moreover, all the indicators and constructs of the survey were reviewed and approved by experts, including all the researchers, the five software professionals, and four academicians involved with managing innovation and software development.

3. Full-scale Survey. After the pilot study, we conducted a full-scale survey. The survey respondents were software professionals from two organizations, with which we had an established relationship.

In order to begin the conduction of the survey, we first presented the research proposal to the contact persons of two study organizations labeled as alpha and beta for purposes of anonymity. The contact persons of these two organizations then forwarded the proposal to related authorities for approval. After the research proposal was approved, the contact persons provided us with a list of names of internal software professionals. From this list of names, we randomly chose 85 people as survey respondents and sent them email invitations to complete the survey.

We received 60 responses. The return rate was about 70 percent, which is relatively high compared to most surveys in the IS field. We think that the high return rate is due to the strong support of the company leaders and contact persons to this research project. Among the 60 responses, three of them are missing a significant amount of data. We hence excluded these three responses from analysis, leaving a sample of 57 usable responses (N=57). This sample size ensures a power level of 0.82.

When we asked the contact persons about unreturned surveys, they replied that they believed that the non-responders were too busy to fill out the survey. Since we do not have the access to non-responders, we know very little about their demographics. This is a caveat of the study related to the sample.

THE SURVEY RESULTS AND FINDINGS

Demographic Information

Analyzing the demographic data, we found that 11 percent of respondents are between the ages 20-29; 67 percent of respondents are between the ages 30-49; 22 percent of respondents are older than 50. In addition, we found that 81 percent of the respondents are male while 19 percent of them are female. We also learned that most respondents have earned a bachelor degree (50%) or a master's degree (36%), while a small proportion of them have a Ph.D. (4%) or other degree (10%).

Among survey respondents, 18 percent of them have worked at their current job for less than a year, 60 percent of them have worked at their current job for 1 to 10 years; and 22 percent of them have worked at their current job for more than 10 years. In addition, among them, 70 percent have worked in the IT industry for over 10 years, while 30 percent of them have worked in the IT industry for less than 10 years. Among the survey respondents, 4 percent of them have never worked on agile projects; 75 percent of them have worked on 1 to 10 agile projects, while 21 percent of them have worked on more than 10 agile projects. Most survey respondents have gained their knowledge on agile methodology on the job or by reading, while few of the respondents have received training.

From the demographic information, we learned that our survey respondents are from a variety of backgrounds and have different levels of experience with, and knowledge of, agile methodology.

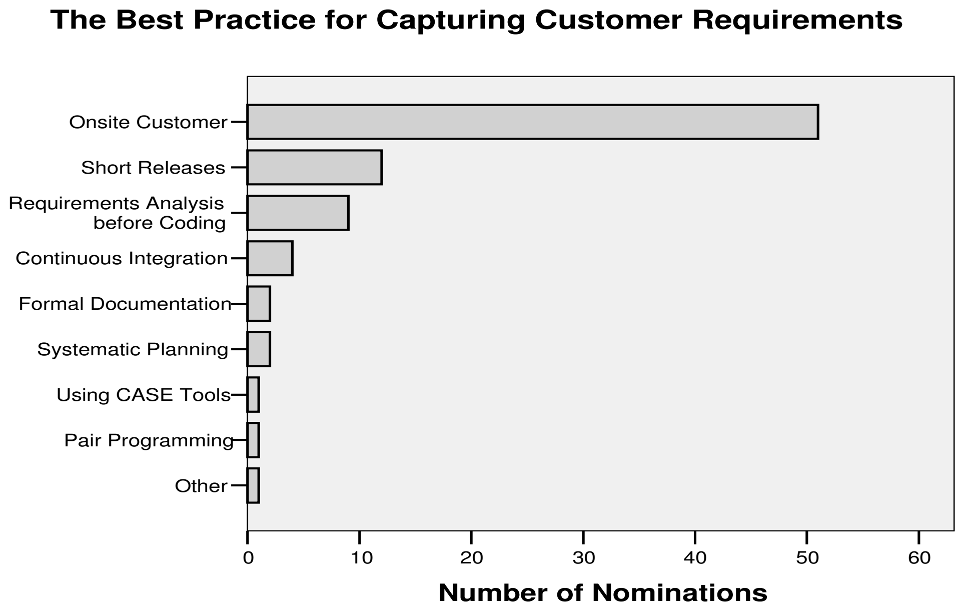

Figure 1 Responses to Best Practices for Capturing Customer Requirements

The Best Practices for Software Quality Improvement

Survey respondents were asked to choose the best practice for various quality improvement purposes in Part I of the survey. The 10 candidate software practices from which they could choose included continuous integration, onsite customer, pair programming, short releases, 40-hour week, formal documentation, process control and improvement, requirements analysis before coding, systematic planning, and using CASE tools. Notice that half of these practices are popular agile practices, such as continuous integration, onsite customer, pair programming, short releases, 40-hour week, while the other half are popular plan-driven practices, such as formal documentation, process control and improvement, requirements analysis before coding, systematic planning, and using CASE tools.

There were seven categories in which respondents were asked to express a preference. For capturing customer requirements, the onsite customer practice was chosen as the most effective practice, while the short releases practice was chosen as the second most preferred practice. These two practices are agile practices, showing that these agile techniques are considered by respondents as more effective for capturing customer requirements than the plan-driven practices. Even though the "requirements analysis before coding practice," a plan-driven practice, took the third place, the number of responses it received is far fewer than that of the onsite customer practice. Figure 1 shows detailed response preferences for development practices.

For assuring that the systems design meets customer needs, survey respondents chose the short releases practice as the most effective practice, while the onsite customer practice, and the continuous integration practice followed. These three practices are agile practices, showing that agile methodology is perceived as having better techniques than plan-driven methodologies in assuring that the systems design meets customer's expectations. We speculate that the more steps and actions a systems development methodology involves, the more likely it is that it decreases perception of design quality and perhaps design quality itself, because of the likelihood miscommunication among various stakeholders will be amplified. Figure 2 shows the distribution of all of the survey responses.

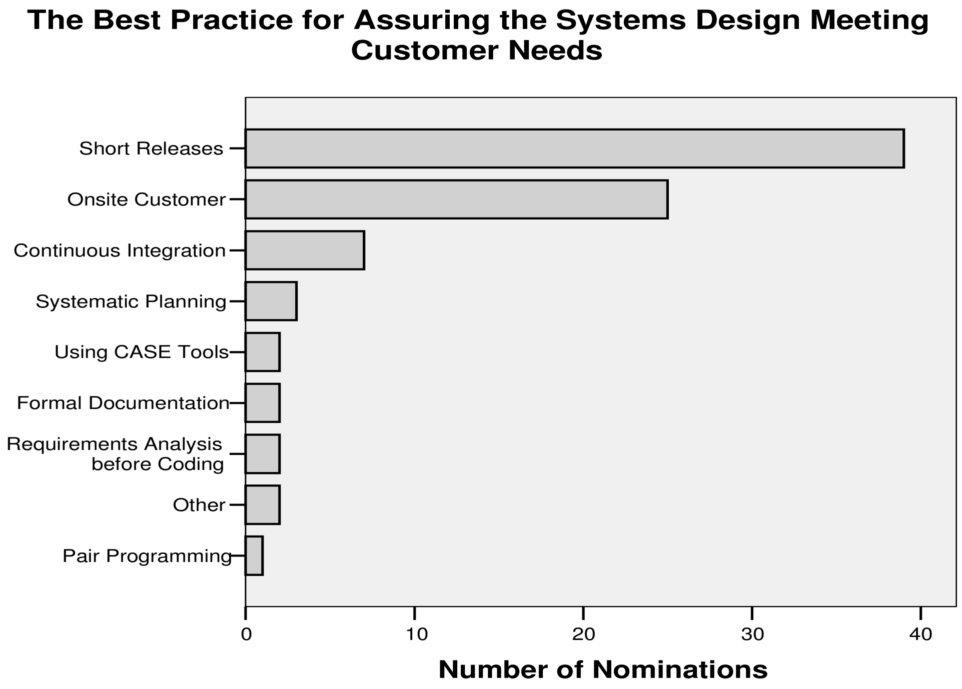

Figure 2 Responses to Best Practice for Assuring that Systems Design Meets Customer Needs

In addition to capturing customer's requirements and designing software that meets customer's needs, eliminating design defects and coding errors were identified by the survey respondents as important for improving software quality. Survey respondents chose the pair programming practice and the continuous integration practice as the most effective practices for eliminating coding errors and design defects. Once again, these two practices are agile practices, underscoring that the survey respondents perceive agile methodologies as being more effective ways than the plan-driven methodologies in eliminating defects. Figure 3 and 4 show detailed response preferences for eliminating coding errors and design defects.

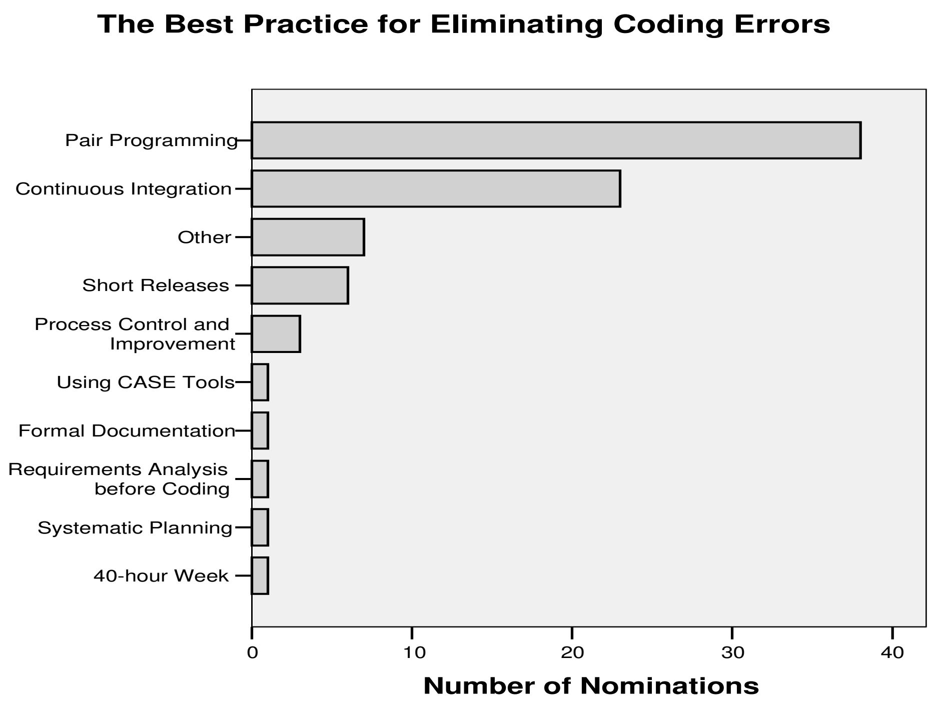

Figure 3 Responses to Best Practice for Eliminating Coding Errors

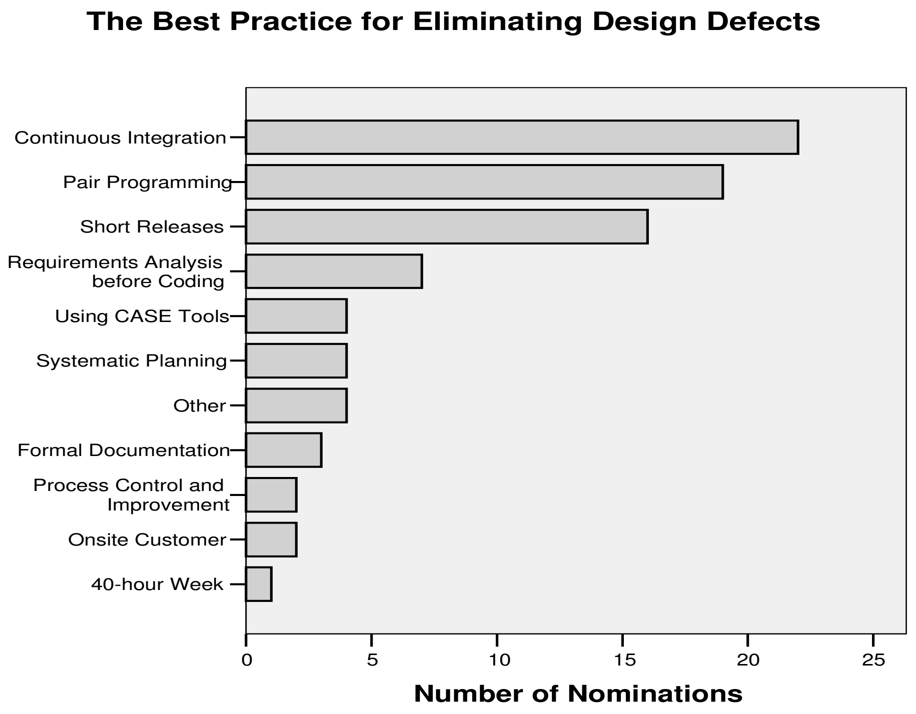

Figure 4 Response to Best Practice for Eliminating Design Defects

Among the 10 software practices, survey respondents chose the short releases practice as the best practice for responding to customers' changing needs, and onsite customer was chosen as the second best practice in this regard. Once again, these two practices are agile practices. This finding supports our initial hunch based on an extensive literature review, that found that many agile methodologists have suggested that the agile methodology is better than the plan-driven methodology in responding to changes. Figure 5 provides the survey responses to the question of best practice for being responsive to customers' needs..

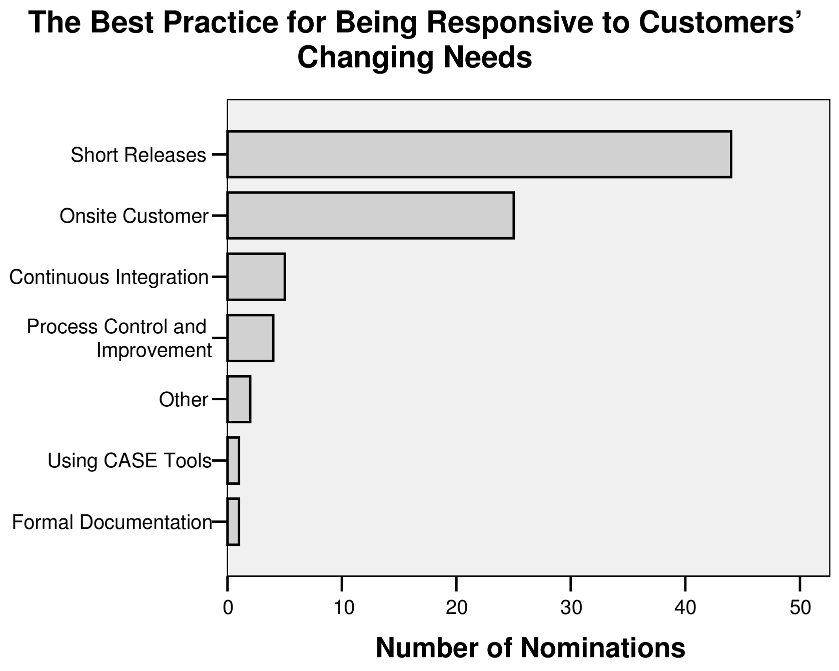

Figure 5 Responses to Best Practice for Being Responsive to Customers' Changing Needs

Among the ten candidate practices survey respondents identified short releases and onsite customer practices as the perceived best practices for producing highly valuable software. Again, this confirms our hunches arising from the literature, since agile practitioners have suggested that agile methodologies are better in producing software of high value (Highsmith, 2002a). Figure 6 provides the detailed breakdown of responses.

Survey respondents identified the short release practice, followed by the onsite customer practice, as the perceived best practice for achieving customer satisfaction. Since these two practices are agile practices, this result supports the perception that agile methodologies have a strong focus on building a long-term customer relationship. By contrast, plan-driven methodologies are considered by many developers as possessing a strong internal focus rather than focusing on achieving customer satisfaction. Software practices of the plan-driven methodologies were consistently placed at the bottom of the ranking by survey respondents. For details, please refer to Figure 7.

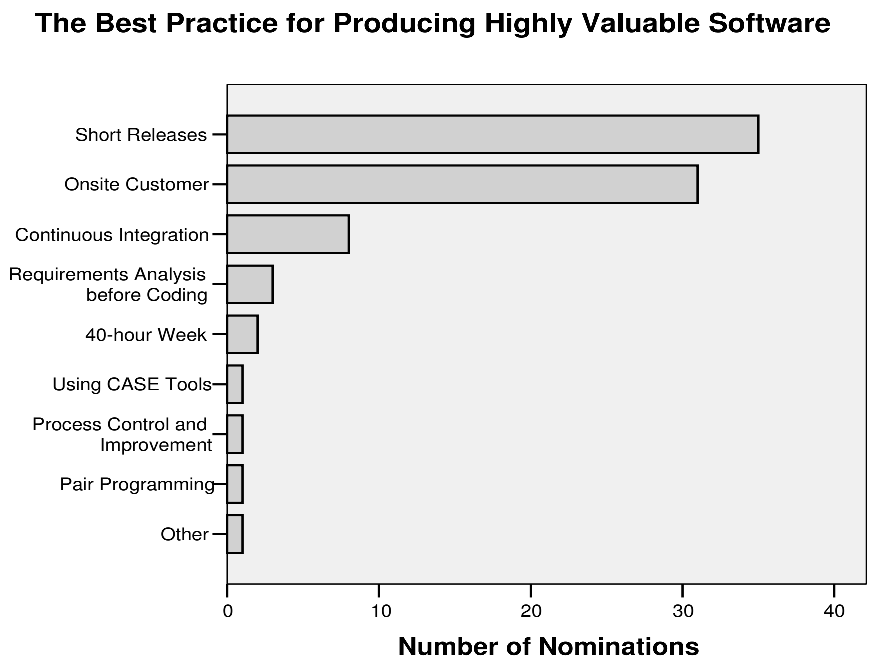

Figure 6 Responses to Best Practice for Producing Highly Valuable Software

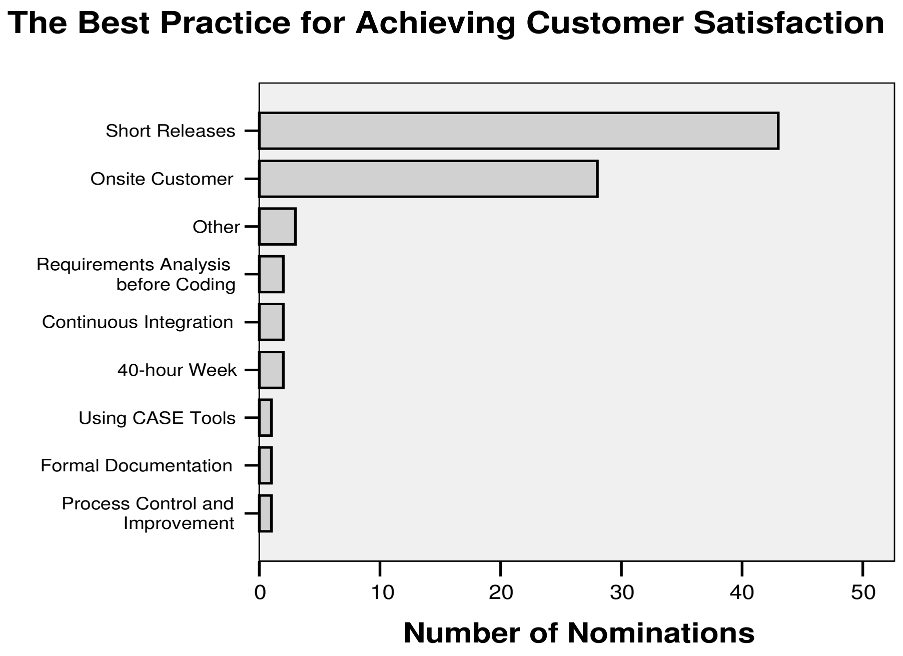

Figure 7 The Best Practice for Achieving Customer Satisfaction

In summary, data analysis shows that software developers consistently perceived agile practices over plan-driven software practices as the most effective way of eliminating coding errors and design defects as well as achieving customer satisfaction. Specifically, we found that onsite customer and short releases were identified as the most effective software practices to capture customer requirements and achieve customer satisfaction; additionally, pair programming and continuous integration are perceived to be the most effective practices for eliminating coding errors and design defects.

Specifically, we found through our survey that: (1) having an onsite customer was identified as the best practice for capturing customer requirements; (2) short releases were chosen as the best practice for assuming a design meeting customer's needs, for being responsive to customers' changing needs, for producing highly valuable results, and for achieving customer satisfaction; (3) pair programming was chosen as the best practice for eliminating coding errors; and (4) continuous integration was chosen as the best practice for eliminating design defects. We see that the agile practices have been consistently identified as the perceived best practices for achieving a variety of quality results. These findings are highly valuable for informing practitioners of the best agile practice for specific quality goals.

CONTRIBUTIONS AND FUTURE RESEARCH

Our paper has made important contributions in the following ways. By differentiating two critical aspects of software quality, the design quality (software design conforms to customer requirements) and the coding quality (the coding conforms to software design), we have achieved an in-depth understanding and accurate findings about the perceived impact of agile practices on software quality.

In addition, in analyzing the perceived impact of agile methodology on these two different quality aspects, we found that agile methodology has significant positive impacts on perceptions of design quality and on the resultant software's coding quality. We have found that agile practices such as onsite customer and short releases are perceived by many developers as better practices in achieving customer satisfaction than plan-driven practices. In addition, we found that agile practices such as continuous integration and pair programming are perceived by many developers as better practices in eliminating coding errors and design defects than are plan-driven practices.

This paper has focused on examining software developers' perspectives or perceptions of the effectiveness of agile practices. The developers appear to see agile methodologies as beneficial, but they are only half of the picture, since the customer and users will be left with the resulting system. Thus, our future study will examine the customers' perspective and perceptions on agile methodology. A survey study targeted to customers and users will be conducted to complete that part of the study. Although this paper only presents the basic descriptive statistics (i.e. what software practices are the most effective in achieving certain quality goals), our next paper will examine the interaction effects.

The participants obviously believe that plan-driven approaches and agile methods are two distinct methodologies. The results of our survey clearly shows that this is their belief. We point out, however, that in practice, a systems analyst or systems development team often uses every tool and trick they know to complete the project. An analyst using SLDC might use agile methods to complete just part of project (for example build prototypes for system output) in order to quickly get reactions from the customer. Likewise, an agile developer may need to rely on data flow diagrams to sort out the differences between logical flow and the physical implementation of the system. We find that these methodologies tend to overlap in practice.

Finally, even though we found that key agile practices were considered to be best practices, we do not advocate that the agile approach is the only systems development methodology or even the best methodology to use in every case. Traditional approaches such as the systems development life cycle might make more sense in many situations, especially when most of the systems have been developed using SDLC in the past. Similarly, object-oriented approaches might be more appropriate if the project lends itself to the discipline, time, and resources required to complete the project.

REFERENCES

1. Ambler, S. (2006). Surveys exploring the current state of information technology practices. Retrieved 2006, 05/01, from http://www.ambysoft.com/surveys/.
2. Armitage, J. (2004). Design: Are agile methods good for design? *Interactions, 11*(1), January 2004.
3. Beck, K. (1999). *Extreme programming explained: Embrace change.* Boston: Addison-Wesley.
4. Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change.* 2nd Edition. Boston: Addison-Wesley.
5. Beck, K., et al. (2001). *Manifesto for agile software development.* Retrieved April 1, 2005, from http://www.agilemanifesto.org.

6. Cockburn, A. (2002). *Agile software development*. Boston: Addison-Wesley.
7. Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer, 34*(11), 131-133.
8. Cohen, J. (1988). Statistical power analysis for the behavioral sciences. NJ: Lawrence Erlbaum Associates.
9. Digital Focus (2005). *Agile 2006 Survey: Results and Analysis.* Retrieved May 1, 2006, from http://www.digitalfocus.com/research/research.php
10. Gryna, F. M., & Juran, J. M. (2001). *Quality planning and analysis: From product development through use* (4th ed.). New York, NY: McGraw-Hill.
11. Highsmith, J. (2002a). *Agile software development ecosystems*. Boston, MA: Addison-Wesley Professional.
12. Highsmith, J. (2002b). What is agile software development? *CrossTalk, The Journal of Defense Software Engineering, Oct. 2002*. Retrieved May 1, 2005, from http://www.stsc.hill.af.mil/crosstalk/2002/10/highsmith.html
13. Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer, 34*(9), 120-127.
14. Huo, M., Verner, J., Zhu, L., & Babar, M. A. (2004). Software quality and agile methods. *Proceedings of the 28th Annual International Computer Software and Applications Conference,* 520-525.
15. Juran, J. M. (1992). Juran on quality by design : The new steps for planning quality into goods and services. New York, NY: Free Press.
16. Moore, G. C., & Benbasat, I. (1991). Development of an instrument to measure the perceived characteristics of adopting an information technology innovation. *Information Systems Research, 2*(3), 192-222.
17. Opperthauser, D. (2003). Defect management in an agile development environment. *CrossTalk, The Journal of Defense Software Engineering, Sep. 2003*. Retrieved May 1, 2005, from http://www.stsc.hill.af.mil/crosstalk/2003/09/0309Opperthauser.html
18. Paulk, M. C. (2001). Extreme programming from a CMM perspective. *IEEE Software, 18*(6), 19-26.
19. Paulk, M. C. (2002). Agile methodologies and process discipline. *CrossTalk, The Journal of Defense Software Engineering, Oct. 2002*. Retrieved May 1, 2004, from http://www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html
20. Schwaber, K., & Beedle, M. (2001). *Agile software development with scrum*. NJ: Prentice Hall.
21. VersionOne & Agile Alliance (2006). *The "State of Agile Development" Survey Results.* Retrieved December 1, 2006, from http://www.versionone.net/surveyresults.asp.