5-2012

Agile Software Development Documentation and Maintainability

Kay M. Nelson
Southern Illinois University Carbondale, ikay@siu.edu

H. James Nelson
Southern Illinois University Carbondale, nelson.j@cob.siu.edu

Benjamin J. R. Wierwille
Southern Illinois University Carbondale, bwierwi@business.siu.edu

Agile Software Development Documentation and Maintainability

Kay M. Nelson
Southern Illinois University Carbondale
ikay@siu.edu

H. James Nelson
Southern Illinois University Carbondale
nelson.j@cob.siu.edu

Benjamin J. R. Wierwille
Southern Illinois University Carbondale
bwierwi@business.siu.edu

ABSTRACT

This research seeks to identify or create best documentation and maintenance practices for Agile software development. Many organizations are attempting to use Agile but problems persist with documentations and maintenance. This is a critical research issue since organizations spend, on average, 70 - 80% of the money in the software development life-cycle on maintenance (Jones, 2000; Jones and Bonsignour, 2011). This research is multi-method using qualitative interviews combined with quantitative surveys (Lee, 1991) to try and determine the documentation and maintainability of Agile projects. The theoretical basis of this research is traditional software engineering, knowledge and expertise, and symbolism and semantics, and it is being performed by a consortium of universities in the United State, Germany, and Brazil.

Keywords

Agile Programming, Software Engineering, Multi-method, Consortium Research.

INTRODUCTION

Agile software development is the latest in a long line of development trends starting with structured programming, progressing through case tools, the move from procedural to object-oriented methodologies, and free open source programming (FOSS), to name a few. All of these are aimed at producing higher quality software in less time and with lower budgets. The birth of Agile is attributed to a group of 17 software developers who met in Utah in 2001 and created the Agile Manifesto (Beck et al., 2001). The twelve basic tenets of Agile development appear quite noble at first glance. They are:

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Close, daily co-operation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

In an era of rapidly changing business models, intense economic pressure, and ever more large and complex systems, these tenets appear to deliver the type of flexibility needed for today's business environment. However, with further scrutiny from the perspective of the software engineer or the MIS professional who looks at things more holistically, these tenets could be cynically looked at as a return to the undisciplined days of "hacking" together systems. This cynicism becomes less "over reaction" and more realistic when the original statement made in the original Agile Manifesto (Beck et al., 2001) is examined:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools

Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more."

It can be argued that all software methodologies value and seek customer collaboration and response to change. It is naïve to assume that in even the most traditional of Waterfall methodologies that the original specification models are not subject to change and customer input as the development progresses. Agile proponents argue that Waterfall-type methods have become overburdened with regimentation, rules, and excessive paperwork and hail Agile as a throwback to the early days of programming where code writing was the main focus of development activity. This is where Agile tenets come precariously close to the "clean-compile and ship" days of disorganized system development that led to systems that were virtually unmaintainable due to lack of documentation. In many cases, even code documentation was not present. The Agile Manifesto actually calls for working software over documentation. This works as long as no changes or fixes are needed after the software is implemented and a maintenance team that had no involvement in development attempts to intuit how the system is meant to work rather than having documentation to refer to. In an era where many systems are being redesigned for customer use, mobile use, or use on devices not yet invented, it is unrealistic to think that maintenance is less important than before. In fact, with Moore's law actually speeding up, existing systems will be subjected to more maintenance rather than less. Software engineering experts have shown that maintenance is 75-80% of total life-cycle costs (Jones, 2000; Boehm, 1987), and that for every step closer to implementation that a defect is found, it costs ten times more to fix it than if it were caught in a previous phase. Since pre-coding phases such as requirements, analysis, and design are minimized in Agile development in an effort to quickly produce working software, combined with minimal attention to documentation, it is not surprising that a Yahoo commissioned study found that six months after implementation of SCRUM systems (the most popular Agile methodology), many were falling apart due to inadequate maintenance support and lack of documentation (Larman and Vodde, 2010).

So why is there so little research on software maintenance? Pragmatically, it is because most IT professionals prefer to work on the latest and greatest technologies and development, whereas maintenance is often seen as boring and less prestigious. In essence, software maintenance is not "sexy". One thing that has been found in academic research is that when projects begin to run over the allotted time and budget, documentation is one of the first things to be dropped. With these universal issues regarding maintenance and documentation, it is clear that Agile projects need to be investigated from a holistic MIS point of view rather than just from a developers perspective.

THEORETICAL LENSES

The goal of this research is not to demonize Agile methodologies, but to determine if the benefits gained in development are retained during maintenance and to try to understand the total system life costs and benefits of using Agile. The research consortium is in a unique position in that we already have both qualitative and quantitative data (Lee, 1991) on over 500 Agile development projects across three countries (Mingers, 2001). These projects can be re-examined through our new theoretical lenses as well as being studied while they are being maintained. We also have access to many additional projects through the United States largest Agile consulting firm. In addition, some of the projects previously studied are being outsourced or offshored (Larman and Vodde, 2010), as well as new projects that have been identified, allowing the researchers to determine if this impacts the benefits and practices of Agile. The following are the theoretical lenses that are being used to evaluate the projects and form hypotheses. The current study phase is the careful theoretical development of propositions and hypotheses using these lenses. This is being done through literature review (Conboy and Fitzgerald, 2004; Dybå and Dingsoyr, 2008; Conboy, 2009; Gwanhoo and Weidong, 2010) re-analysis of existing data, and additional discovery interviews. Once the hypotheses are developed they will be tested both qualitatively and quantitatively. Having a team of five to ten researchers allows a project of this scale to be done rigorously and also allows for researcher cross validation. Even though some of the data has been collected in languages other than English, all of the researchers are fluent in English which will be the primary language of publications.

Knowledge and Expertise Lens

One of the much touted and researched aspects of Agile development techniques is having more knowledgeable development teams by using teams of cross-functional experts. There is no question that having a database expert work on that part of a project should result in a better quality data interface to the project. However, database experts (for example) see the world from the focus of the data and the structure of the database and database management system. Without a clear overall project

model which includes a high level data flow diagram, such as those created by traditional Waterfall methodologies, the potential exists for the database expert to focus on the wrong level of granularity for a single timebox of Agile development. The purpose of detailed dataflow diagrams (DFD) in traditional methodologies is not only to make sure every expert on the team is working from the same blueprint, but to also capture and potentially improve the business process represented by the DFD. Although Agile project teams are supposed to meet for brief periods on a daily basis, experts from different areas that are team members may believe they are clearly communicating to each other when in fact they are leaving out crucial details that they take for granted because of their tacit expert knowledge (Nelson, Nadkarni, Narayanan, and Ghods, 2000). Experts have a very difficult time expressing tacit knowledge that is second nature to them, and it is unlikely that this important knowledge would come out in a brief meeting or in minimal documentation. This may be one of the reasons that Agile teams often fail to complete requirements assigned to a particular timebox and the requirements are put back into the general requirements pool. While there is data to suggest that Agile team members do learn from each other, there has been no study that has compared the cognitive grasp and cohesion or dissonance of the scope and goals of a project between Agile and other methodologies. Agile proponents argue that software development is fluid and change is a natural occurrence, but there is little evidence that these properties are not addressed in other methodologies. It appears that the major difference is that a Waterfall methodology starts with high-level detailed documentation that all experts "buy in" to, and as changes occur during development, that documentation is modified. With no high-level documentation in Agile methodologies, it is possible that whatever documentation that is created by individual experts may be in semantics not clear to someone with different expertise or to those in charge of maintaining the system (Turk, France, and Bernhard, 2002). While studies have shown that Agile teams do learn from each other, it is not known how much of another experts knowledge frame is absorbed (Mathieu, Goodwin, Heffner, Salas, and Cannon-Bowers, 2000). A key benefit of Agile is having a user "owner" as a full member of the team, but no research has indicated whether this business person fully understands the technical interchanges in the daily Agile briefings.

Symbolism and Semantic Lens

Hirshheim and Newman wrote a seminal paper in 1991 describing the software development process as myth, metaphor, and magic versus the rational economic undertaken it is assumed to be. They contend that the actual focus of the process is symbolism and that information systems development (ISD) is really about interpreting social interactions versus following regimentation and rules. The Agile Manifesto appears to support this view, emphasizing individuals and interaction as well as team collaboration. Hirshheim and Newman (1991) warn that developers are usually poorly trained to fully embrace these social processes and propose these symbols of myth, metaphor, and magic. The myths described actually match the Agile Manifesto quite well with the exception of "*The Key to Successful Design Is the Use of a Top-Down Approach*" (pg 36). This myth also involves the use of the common semantics in high level design documentation that appears to be missing with the Agile approach. Hirshheim and Newman's case study actually shows that the lack of a top-down approach resulted in the abandonment of a $2 million dollar project.

Pondy (1983) describes metaphors as having the dual purpose of facilitating change while maintaining stability. While Agile emphasizes change facilitation, it may be misleading for the Agile team to believe they are in complete control of the project. Pondy (1983, p. 164) states, "In organizing, the use of metaphor *simultaneously* facilitates change *and* reinforces traditional values . . . metaphor can fulfill the dual function of enabling change and preserving continuity." The danger with Agile timeboxes and sprints to catch up with requirements not completed in timeboxes, may give the team the false security that they are delivering everything necessary to maintain the system (Klimoski and Mohammed, 1994). Ironically, the most popular Agile methodology, SCRUM, uses the game of rugby as a metaphor for how development is accomplished.

Magic is probably the strongest symbol in the Agile approach. Members of Agile teams are true believers and discount the efficacy of other approaches. The actual production of a manifesto is the very manifestation of magical thinking that Hirshheim and Newman describe. However Agile does attempt to nullify one type of negative magic that the authors found in their case studies; that user involvement is a façade. By having a user representative at every team meeting Agile should promote real user involvement (Balijepally, Mahapatra, Nerur, and Price, 2009). By looking for evidence of these symbols and semantics in our dataset and in future maintenance studies, we believe that symbolism and semantics can give us a rich understanding of the actual workings of Agile throughout the life of a system.

Software Engineering Lens

Shortly after the Agile Manifesto was published, notable software engineers such as Barry Boehm (2002), Barbara Kitchenham (Kitchenham, Pfleeger, Pickard, Jones, Hoaglin, El Emam, and Rosenberg, 2002), Dieter Rombach (Rombach,

Schneider, Kitchenham, Pfahl, and Selby, 2007), and Kurt Schneider (Rombach et al., 2002) began examining the methodology to see if software quality metrics could be applied. While they recommended adding technical writers to Agile teams and quality assurance testing for documentation consistency and quality, no literature exists that demonstrates this has been done. Agile development has been approached from the Software Engineering Institute's Capability Maturity Model (Kussmaul and Jack, 2007; Jakobsen and Johnson, 2008), but yet again, ongoing maintenance of the system has not been addressed. The literature on software engineering has attempted to introduce "process" into Agile development (Karlström, and Runeson, 2006), yet the basic premise of Agile is to avoid formal process as much as possible (Symons, 2010). One major research question that needs to be answered is whether Agile and Software Engineering are conflicting paradigms (Turner and Boehm, 2003; Gotel, 2011).

A Possible Fourth Lens

Throughout the other three lenses is a consistent theme of little or no high-level design documentation which strongly suggests that lack of architecture and conceptual modeling could be a major factor in the maintainability of Agile systems. If this is indeed a fourth lens, it would be examined from a design science perspective.

DISCUSSION

Agile development may be here to stay or may go the way of the much touted CASE tools – millions of dollars invested and when looked at from a total system life perspective, discarded. Our initial data shows that some larger Agile development projects are beginning in a more traditional manner with DFDs or in the case of object-oriented systems, class libraries and use cases (Leffingwell, 2007). The goal of our research is to evaluate Agile projects that have been operational for a year or longer, and to understand the maintainability of these systems and discover what documentation is being used in the maintenance process. By doing this, we believe we can discover the best and worst practices used in Agile development, and give organizations clear criteria to decide if Agile development is the right approach at the right time. Unfortunately, like many development methods (and technologies) before it, Agile development is a trend that many organizations decide to adopt because it is the "hot" new thing to do. As scientists, we are examining a very large number of Agile development projects across three continents. We are using multiple lenses to view this phenomenon and multiple methods for analysis to try and discover how to best spend the 75-80% of the total system cost that is in maintenance (Jones and Bonsignour, 2011).

REFERENCES

1. Balijepally, V., Mahapatra, R, Nerur, S. P., and Price, K. H. (2009) Are two heads better than one for software development? The productivity paradox of pair programming, *MIS Quarterly*, 33, 1, 91–118.
2. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001) Manifesto for Agile Software Development, http://agilemanifesto.org. Accessed 2/13/2012
3. Boehm, B. (2002) Get Ready for Agile Methods, with Care. *IEEE Computer* 35, 1 64–69.
4. Boehm, B. W. (1987) Improving software productivity, *IEEE Computer*, 20, 9, 43-57.
5. Conboy, K. (2009) Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research* 20, 3, 329–354.
6. Conboy, K., and Fitzgerald, B. (2004) Toward a Conceptual Framework of Agile Methods," in *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*,Newport Beach, CA, November 5, 2004, pp. 37-44.
7. Dybå, T. and Dingsoyr, T (2008) Empirical studies of agile software development: A systematic review, *Information and Software Technology*, 50, 9-10, 833–859.
8. Gotel, O. (2011) Requirements Tracery. *IEEE Software*, 28,5,92-94
9. Gwanhoo, L. and Weidong, X. (2010) Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. *MIS Quarterly* 34, 1, 87-114.
10. Hirschheim, R. and Newman, M. (1991) Symbolism and Information Systems Development: Myth, Metaphor, and Magic *Information Systems Research* 2 1 29-92.
11. Jakobsen, C. and Johnson, K. (2008) Mature Agile with a Twist of CMMI, Proceedings of Agile 2008, August 4-8 Toronto. 212-217.

12. Jones, C. (2000) Software Assessments, Benchmarks, and Best Practices, Addison-Wesley Professional, Upper Saddle River, NJ.
13. Jones, C. and Bonsignour, O. (2011) The Economics of Software Quality, Addison-Wesley Professional, Upper Saddle River, NJ.
14. Karlström, D. and Runeson, P. (2006) Integrating agile software development into stage-gate managed product development," *Empirical Software Engineering*, 11, 203–225.
15. Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J. (2002) Preliminary guidelines for empirical research in software engineering, *Software Engineering, IEEE Transactions on Software Engineering*, 28, 8, 721–734.
16. Klimoski, R. and Mohammed, S. (1994) Team Mental Model: Construct or Metaphor," *Journal of Management* (20:2), pp. 403-437.
17. Kussmaul, C., and Jack, R. (2007) Agile & CMM: Worlds Apart, or Best of Both? Proceedings of the SPIN Meeting, January 23rd, Philadelphia
18. Larman, C. and Vodde, B. (2010) *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*, Addison-Wesley Professional, Upper Saddle River, NJ.
19. Lee, A. (1991) Integrating Positivist and Interpretive Approaches to Organizational Research," *Organization Science* (2:4), pp. 342-365.
20. Leffingwell, D. (2007) *Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series)*. Addison-Wesley Professional, Upper Saddle River, NJ.
21. Mathieu, J., Goodwin, G., Heffner, T., Salas, E., and Cannon-Bowers, J. A. (2000) The Influence of Shared Mental Models on Team Process and Performance, *Journal of Applied Psychology*, 85, 2, 273-283.
22. Mingers, J. (2001) Combining IS Research Methods: Towards a Pluralist Methodology," *Information Systems Research*, 12, 3, 240-259.
23. Nelson, K, Nadkarni, S., Narayanan, V., and Ghods, M. (2000) Understanding Software Operations Support Expertise: A Revealed Causal Mapping Approach, MIS Quarterly, 24, 3, 475-507.
24. Pondy, L., Frost, P., Morgan, G. and Dandridge, T (1983) *Organizational Symbolism*, JAI Press, Greenwich.
25. Rombach, D., Schneider, K., Kitchenham, B., Pfahl, D., and Selby, R. (2007) *Empirical Software Engineering Issues, Critical Assessment and Future Directions: International Workshop*, June 26-30, Dagstuhl Castle, Germany.
26. Symons, C. (2010) Software industry performance: What you measure is what you get, *IEEE Software*, 27, 6, 66-72.
27. Turk, D., France, R. and Bernhard, R. (2002) Limitations of agile software processes. *Proceedigns of the 3rd International Conference on eXtereme Agile Processes Software Engineering*. Alghero, Sardinia, Italy.
28. Turner, R. and Boehm, B. (2003) *Balancing Agility and Discipline: A Guide for the Perplexed Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, Upper Saddle River, NJ.