Requirements Specifications on Agile Projects

Business analysis is an important aspect of agile software development projects, but the agile approach is significantly different than the traditional, serial approach of yesteryear. Because the agile approach to business analysis is different the approach to requirements specification is also different, for many traditionalists this will prove to be a significant cultural shock to them at first. In this article I briefly overview how business analysis activities fit into an agile approach, question some of the dogma around documentation within the traditional community, summarize some of the evidence showing that agile approaches are more effective in practice than traditional approaches, and end with strategies for specifying requirements on an agile project.

The Agile Approach

Business analysis is so important to agile development projects that we're prepared to do it every day throughout the lifecycle, it's not just a phase that we go through early in the project. Agile Model Driven Development (AMDD) includes several best practices [1] which are pertinent to understanding how business analysis fits into an agile project. These practices are:

- Active stakeholder participation. Stakeholders should provide information in a timely manner, make decisions in a timely manner, and be as actively involved in the development process through the use of inclusive tools and techniques.
- Prioritized requirements. Agile teams implement requirements in priority order, as defined by their stakeholders, so as to provide the greatest return on investment (ROI) possible.
- Model a bit ahead. Sometimes requirements that are nearing the top of your priority stack are fairly complex, motivating you to invest some effort to explore them before they're popped off the top of the work item stack so as to reduce overall risk. This should be a rare event, but does occur from time to time.
- Requirements envisioning. At the beginning of an agile project you will need to invest some time to identify the scope of the project and to create the initial prioritized stack of requirements. This effort should take a few days, up to two weeks, assuming you can overcome the logistical challenges associated with getting the right people involved.
- Iteration modeling. At the beginning of each iteration you will do a bit of modeling as part of your iteration planning activities.
- Model storming. Throughout an iteration you will model storm on a just-in-time (JIT) basis for a few minutes to explore the details behind a requirement or to think through a design issue.
- Test-driven development (TDD). Write a single test, either at the requirements or design level, and then just enough code to fulfill that test. TDD is a JIT approach to detailed requirements specification and a confirmatory approach to testing.

AMDD promotes an approach to analysis activities which is a highly iterative, highly collaborative, and very flexible while still addressing the inherent risks associated with requirements. An interesting philosophy within the agile community is that a changed or new requirement late in the lifecycle can be turned into a competitive advantage if you're able to readily act on it. The agile approach to requirements and analysis activities can be very different, and uncomfortable at first, to people with more traditional backgrounds.

Documentation Dogma

Through years of hard-earned experience agilists have come to question some of the traditional dogma surrounding the value of documentation. First, agilists observe that a "big requirements up front (BRUF)" approach where you create a detailed requirements document early in the project lifecycle proves to increase overall project risk and results in significant wastage in practice [2]. There are several reasons for this, including people's difficulty in defining what they actually need, documentation being an incredibly poor way to communicate information (more on this in a minute), and traditional "change prevention" strategies [4] motivating stakeholders to stuff requirements specifications with fictional requirements. Agilists recognize that the real goal isn't to write a detailed requirements specification but instead is to understand the intent of your stakeholders and then produce a solution that addresses that intent effectively. To accomplish this agile development teams embrace change, accepting that requirements will change over time as stakeholders learn based on the regular feedback acquired from the regular production of potentially shippable software.

Second, agilists have accepted the lessons of media richness theory (MRT) which shows that documentation is the worst strategy available to communicate information between people [4]. One of the many things which comes from MRT is an examination of various strategies for communicating information, which in turn provides insight into the risks that you take on as the result of process-related decisions. In the Autumn of 2008 I decided to validate the lessons from MRT for agile software development teams through a survey that I ran on the Extreme Programming, Scrum Development, Test-Driven Development (TDD) and Agile Modeling mailing lists. Two of the questions explored the effectiveness of communication strategies between developers within a team and between team members and stakeholders. The results are summarized in **Table 1**, with answers rated on a range of **-5 (very ineffective)** to **+5 (very effective)**, aligning fairly well with what MRT predicted. It was interesting to note that overview documentation was perceived as being reasonably effective although detailed documentation was not. Also, online chat was thought to be effective between developers but not with stakeholders, likely a reflection of cultural differences and experiences between the two communities.

Table 1. Effectiveness of communication strategies on agile development teams.

Communication Strategy	Within Team	With Stakeholders
Face to face (F2F)	4.25	4.06
F2F at Whiteboard	4.24	3.46
Overview diagrams	2.54	1.89
Online chat	2.1	0.15
Overview documentation	1.84	1.86
Teleconference calls	1.42	1.51
Videoconferencing	1.34	1.62
Email	1.08	1.32
Detailed Documentation	-0.34	0.16

A serious cultural challenge which traditionalists will struggle with is the belief in the value of specialization, in this case in the value of someone specializing on being an analyst. Although there is significant value in analysis activities it doesn't imply that there's value in someone doing only that. The problem with someone who only knows how to model and document, then that's what they're going to do whether that's the best option or not. But, if they have a wider range of skills, such as testing and programming skills, then they have a wider range of options available to them and are much more likely to use the most appropriate technique for the situation. Agile teams are finding that people with one or more specialties, you've got to be able to add value, AND at least a general knowledge of the software process and the domain that they're working in, are much more effective in practice than people who are just specialists. These people are referred to as generalizing specialists [6].

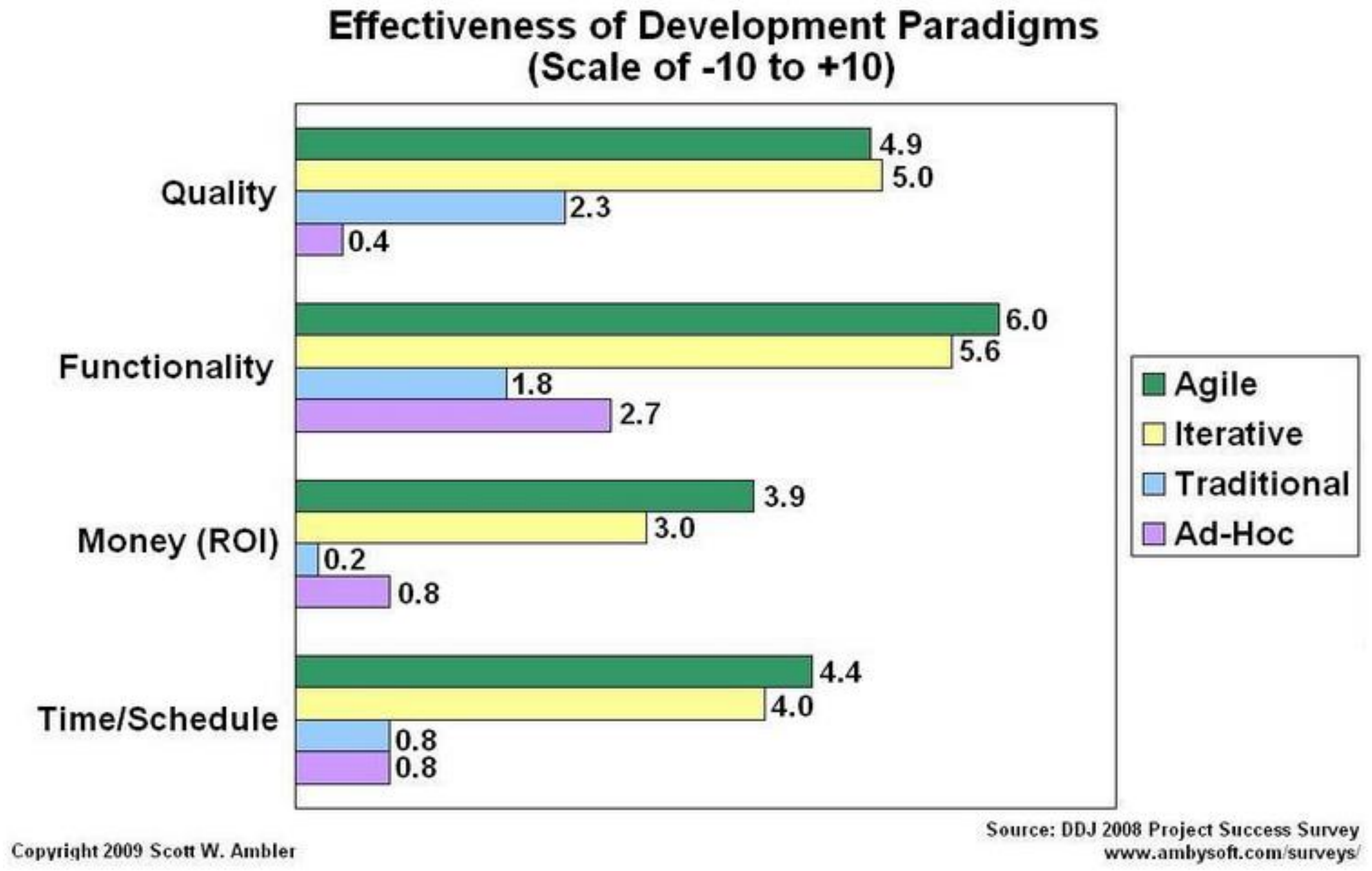
And Agile is Working Better

In early December 2008 I sent a survey out to the Dr. Dobb's mailing list, a group of people working in a wide variety of situations and following a range of different strategies, to explore actual project success rates [7].

Unlike other research which forces a single definition of success on people we explored both how people define success, discovering a wide range of responses, and allowed them to define success on their own terms (something that they do in reality). We found that project teams following an iterative process such as the Unified Process (UP) did best with an average 71% success rate. Second best were agile project teams, perhaps following Scrum or Extreme Programming (XP), with an average 70% success rate (statistically the same result as iterative teams). Third place were traditional project teams following a serial approach to development, perhaps based on the V model, with an average 66% success rate. Finally, ad-hoc teams which didn't follow a defined process had a success rate of 62%.

Although the success rates are interesting, what's more interesting is how each paradigm rates on critical success factors. Figure 1 depicts this, showing the results on a scale of -10 (very unlikely) to +10 (very likely) for the ability of teams following each paradigm to deliver high-quality systems, to deliver functionality that stakeholders actually need, to produce good return on investment (ROI), and to deliver the system in a timely manner. As you can see both iterative and agile approaches clearly outshone traditional strategies. What should be of interest to the analyst community is that agile teams, for whatever reasons, appear to be more effective at addressing stakeholder needs than traditional approaches. The implication is that although traditionalists might not like the agile approach they should at least sit up and take notice of them. Furthermore, one reason why the traditional success rate was almost as high as the agile success rate may be because people have lower expectations of traditional teams (traditional teams clearly aren't delivering as effectively as agile teams) - one of the problems with letting respondents define success in their own terms is that they may not be using the same definitions of success for different types of projects.

Figure 1. Comparing effectiveness of development paradigms.



How Much Requirements Specification?
There is a fair bit of confusion amongst the analyst community surround requirements specification on agile software development projects. Some of that is a lack of understanding of agile approaches to modeling and documentation, some of is the result of having the traditional dogma around documentation being inflicted upon them for decades, and some of it is the result of the agile community's focus on relatively simple "level 1" environments. At IBM we've been working on an Agile Process Maturity Model (APMM) [8], the goal of which is to help put agile processes and practices into context -- unlike the staged approach to the Software Engineering Institute (SEI)'s Capability Maturity Model Integrated (CMMI) which is to provide a rating system for your software process. Anyway, there are three levels to the APMM: level 1 processes such as Scrum which focus on a portion of the software development process, level 2 processes such as Unified Process (UP) and Dynamic System Development Method (DSDM) address the full delivery lifecycle, and level 3 processes are effectively level 2 processes modified to address one or more scaling factors. Distinguishing between these levels can help you to recognize when you need to adopt various specification strategies, or not adopt them as the case may be.

Let's explore some agile strategies for documentation and specification [9] and see how to apply them at various levels of scale. In addition to the modeling strategies list above, AMDD also includes several documentation-oriented strategies:

- Just barely good enough (JBGE) artifacts. A model or document needs to be sufficient for the situation at hand and no more. This is an important throttle on much of the out-of-control bureaucracy exhibited by many traditional teams. However, JBGE is situationally dependent, making it difficult to prescribe exactly how much to specify, when to do it, and in what format.

- Document late. Write documentation as late as possible, avoiding speculative ideas that are likely to change in favor of stable information. In other words, early investment in specifications will likely prove to be a waste due to the speculative nature of requirements definition.

- Executable specifications. If requirements should be testable, why not skip over the documentation middleman and specify requirements in the form of executable "story tests"? Why not capture your design details as executable developer tests, instead of non-executable "static" documentation?

- The mainstream, level 1, agile advice often focuses on small, co-located teams. Although these situations are in fact very common, it's been estimated that 75% of all software development teams regardless of paradigm are ten people or less, the reality is that many teams find themselves outside of this zone. As the situation that you find yourself in becomes more complex, your strategy for addressing the inherent risks must change. Let's explore how each of the six agile scaling factors [10] can affect your approach to requirements elicitation and specification:

- Team size. Larger teams will often address more complex problems, requiring greater coordination of requirements. Large teams are often organized as teams of teams, and each individual subteam will have its own product owner and work item list. The requirements on these list must first be identified and allocated to the various stacks, and then throughout the project new requirements, changing requirements, and requirements dependencies must be coordinated via a product ownership team [11]. Because greater coordination is required there potentially will be greater need for requirements overview documentation, some form of traceability matrix to manage dependencies between requirements, and electronic capture of the requirements (index cards generally don't work well on teams of several hundred people).

- Geographical distribution. Many agile teams are distributed - even if people are working in different cubicles that's a form of distribution let alone if they're working from home or from a different city. As with large teams there will be a greater need for coordination, potentially requiring a bit more specification and a move to electronic capture.

- Regulatory compliance. When regulatory issues - such as Sarbanes Oxley, ISO 9000, or FDA CFR 21 - are applicable there is generally a greater need for documentation. However, read the regulations first. Rarely do they require the mounds of documentation, onerous reviews, and comprehensive traceability matrices which the bureaucrats among us perceive are needed. An important observation is that if you let bureaucrats define your process you'll end up with a bureaucratic strategy, but if you let practice people define it you'll end up with a very pragmatic strategy.

- Enterprise discipline. Enterprise disciplines such as enterprise business modeling [12], can both increase and decrease the need for requirements specification. There will likely be an increase in the need to maintain traceability back to your enterprise models but a decrease in the need to capture common terminology, business entities, business rules, and so on. Futhermore, the enterprise business modelers are important stakeholders who can be valuable resources for product owners and are in effect enterprise product owners in their own right.

- Organization distribution. Sometimes a project team includes members from different divisions, different partner companies, or from external services firms. This lack of organizational cohesion increases project risk. The traditional strategy is to put a "documentation band-aid" over these risks whereas the agile strategy is to increase collaboration and reduce the feedback cycle through short iterations. Yes, there will likely be a need for contracts between the organizations. And if some of the organizations aren't working in an agile manner there will be a need for more documentation (or better yet the need to help them adopt an agile approach).

- Application complexity. Some applications are more complex than others, particularly when multiple platforms and legacy systems and data sources are involved. The traditional strategy is to try to fully explore and document the "as is" environment before proceeding, this is particularly true when multiple legacy data sources are involved, but in practice this strategy reflects the working preferences of the team and not the inherent complexities of the problem domain. Any supporting documentation regarding legacy systems, if it doesn't already exist, can be developed on an iterative, as-needed manner via a JBGE strategy.

Parting Thoughts

The critical thing to remember is that there are no hard-and-fast rules for what to specify or in what detail requirements specifications, if any, should be written. This is determined by the situation, and every team will find itself in a unique situation which brings different factors into play. Agilists will write specifications which are just barely good enough (JBGE) for the situation, will do so in an iterative manner, and prefer executable specifications (working tests) over static documents. These agile approaches to requirements elicitation and capture, and to system delivery in general, require greater discipline on the part of practitioners as well as new skills.

An important point is that although the scaling factors can sometimes dramatically increase the desire of people to have more detailed requirements specifications, often early in the lifecycle, the actual need for such documentation increases much less than the desire. A good rule of thumb is that you will need significantly less documentation than many traditional analysts believe but a bit more than what mainstream agile developers are willing to admit to. Err on the side of caution and write less than you think, if you find that you need more documentation then write it at that time.

References

1. Agile Modeling Best Practices. www.agilemodeling.com/essays/bestPractices.htm

2. Examining the "Big Requirements Up-Front (BRUF)" Approach. www.agilemodeling.com/essays/examiningBRUF.htm

3. The "Change Prevention Process" Anti-Pattern. www.ambysoft.com/essays/changePrevention.html

4. Communication on Agile Projects. www.agilemodeling.com/essays/communication.htm

5. Agile Practices and Principles Survey. www.ambysoft.com/surveys/practicesPrinciples2008.html

6. Generalizing Specialists: Improving Your IT Skills. www.agilemodeling.com/essays/generalizingSpecialists.htm

7. DDJ 2008 Project Success Survey. www.ambysoft.com/surveys/success2008.html

8. The Agile Process Maturity Model (APMM). www.ibm.com/developerworks/blogs/page/ambler?entry=apmm_overview

9. Agile/Lean Documentation: Strategies for Agile Software Development. www.agilemodeling.com/essays/agileDocumentation.htm

10. Agile Scaling Factors. www.ibm.com/developerworks/blogs/page/ambler?entry=agile_scaling_factors

11. Roles on Agile Teams: From Small to Large Teams. www.ambysoft.com/essays/agileRoles.html

12. The Enterprise Business Modeling Discipline: Scaling Agile Software Development. www.enterpriseunifiedprocess.com/essays/enterpriseBusinessModeling.html

---

Author: "**Scott W. Ambler** is Chief Methodologist/Agile with IBM Software Group, working with IBM customers around the world to help them to improve their software processes. He is the founder of the Agile Modeling (AM), Agile Data (AD), Agile Unified Process (AUP), and Enterprise Unified Process (EUP) methodologies. Scott is the (co-)author of 19 books, including Refactoring Databases, Agile Modeling, Agile Database Techniques, The Object Primer 3rd Edition, and The Enterprise Unified Process. Scott is a senior contributing editor with Information Week. His personal home page is

http://www.ibm.com/software/rational/leadership/leaders/#scott

and his Agile at Scale blog is

www.ibm.com/developerworks/blogs/page/ambler

---

http://www.modernanalyst.com/Resources/Articles/tabid/115/ID/923/Requirements-Specifications-on-Agile-Projects.aspx