Extending Agile Methods: A Distributed Project and Organizational Improvement Perspective

Paul E. McMahon
PEM Systems

Tuesday, 19 April 2005
Track 7: 4:50 – 5:35 p.m.
Room 251 A-C

It has been argued that agile methods only work for small, collocated, self-directed teams that include on-site customers. But what if your customer cannot be on-site full-time, or your development team is distributed around the world, or your developers lack self-directed team skills? Does this mean you cannot take advantage of agile methods? This article presents a case for using key agile practices along with recommended extensions on a broader range of projects, including large and physically distributed efforts. The article motivates the use of agile methods by exposing common myths and providing information that can help managers and customers facilitate practical agility within their organizations.

Many organizations are looking seriously at agile methods to see if there are benefits to be gained across a broader range of projects. Unfortunately, today these methods are often misunderstood and misapplied. To aid understanding, let us start with some fundamentals and misperceptions.

The Agile Manifesto, which was put together by the founders of many of the most popular agile methods, contains four value statements:

- We value individuals and interactions over processes and tools.
- We value working software over documentation.
- We value customer collaboration over contract negotiation.
- We value responding to change over following a plan. [1]

A key agile value is customer collaboration. This value can be observed through User Stories [2] that are employed by eXtreme Programming (XP) for requirements. User Stories are intentionally high-level with details worked out collaboratively and informally between developer and customer. Nevertheless, I have heard a customer say, "I do not see how agile can help, because I need all my requirements." The belief that agile means that customers must live without all requirements is a misperception.

Another common agile value is working software demonstrated to customers often through short development iterations. The motivation for this value is the belief that a better way to ensure customer needs are met is through working software rather than through formal written words.

In response, I have heard customers and contractors say the following:

- "The lack of up-front planning and requirements on agile projects leads to chaos."
- "Short iterations do not work on large projects because there isn't time to get the design done."
- "Short iterations on complex projects lead to team burnout."

To motivate and provide greater insight, let us now discuss six agile myths.

Myth 1: Agile Methods Do Not Include Plans and Requirements

Many who claim to be agile are in fact using a code-and-fix approach. Agile is not code and fix. It is both incremental and iterative; however, its iterative aspect is often misunderstood. Iterative means that inside each increment there are smaller cycles of development occurring (each usually from four to 12 weeks).

The important point often missed is that each iteration is not just code, but includes plans, requirements, design, code, and test. Those familiar primarily with traditional development methods used on large projects often do not understand how this is possible. The key to agile lies in the fact that the activities, their sequence, and the resultant agile artifacts are not traditional.

If you are a customer or manager familiar only with traditional development methods and you want to take advantage of agile methods, then you will want to know how to distinguish agility from code and fix. Understanding a few practical rules can help.

Practical Rule 1: Every Agile Iteration Is Planned and Measured Regardless of Iteration Length [3]

Agile projects use a two-tier approach to plans: a long-term, coarse-grained plan, and a short-term, fine-grained plan. [4]. On agile projects, planning occurs continually to ensure the team is always focused on *the most important things now*. I was asked by one client, "How can I tell if my team is really doing agile planning, or just reacting to the next fire?" Asking your team the following two questions can help.

- **Question 1:** "How do you determine the most important things?" After you ask this question, listen for the word *risk*. If your team is really doing agile planning, you should hear how the risks perceived by both the development team and the customer are being handled collaboratively. Teams that are reactive often do not take time to collaborate.
- **Question 2:** "How do you reflect the results of your continual planning?" If you hear, "We are agile so we do not document our plans," then your team is not agile. Contrary to what you might have heard, agile teams do document their plans, but the resultant planning artifacts look different. Examples of agile planning artifacts are allocation of user stories to iterations, and task sign-up sheets.

Myth 2: Agile Methods Do Not Allow Requirements Control

Agile methods do not guarantee requirements control, but they do allow it. One way requirements can be controlled with agile methods is to use two levels of requirements. The first level is the high-level User Stories that scope the *complete* project. This level includes features that have not yet been fully analyzed. In Scrum this potential work is placed on what is referred to as the Product Backlog.

The second level of requirements is developed collaboratively with the customer and establishes in greater detail and clarity the work and priorities for the next iteration. In Scrum this work is referred to as the Sprint Backlog.

Not long ago, the company president of one of my agile clients was having a beer with one of his customers. As the

customer was explaining a desired feature, the president looked the customer in the eye and said, "We can do that." A few weeks later the customer was observing a demonstration of the current iteration software and became upset because he did not see the desired feature.

It is worth noting that the customer's desired feature referred to in this story may or may not be within the scope of the current requirements. To ensure requirements are controlled, I recommend that customer requests such as these be first placed on the Product Backlog. Once this potential work is analyzed, clarified, and approved, it may then be placed on the Sprint Backlog. The Sprint Backlog is fixed at the start of each iteration.

The point of the story is simple. Customer collaboration does not mean the contractor must implement everything the customer asks for. With agile, there can still be out-of-scope requests. The agreed-to detailed requirements are established collaboratively by the customer and contractor for each iteration.

Practical Rule 2: The Work for Each Agile Iteration Is Fixed At the Start of Each Iteration [3]

While these practical rules may seem obvious, do not dismiss them lightly. With agile methods we have few prescriptive rules – and this can aid team productivity – but only if those few rules are consistently followed to keep the agile team from falling into chaos.

It is worth noting here that Alistair Cockburn pointed out to me that while this type of rule is a good starting place, some advanced agile groups allow more dynamic changes to the content of an iteration. This has been referred to as Dynamic Scrum [5].

Myth 3: The Schedule Never Slips With Agile Methods

While it is true that with agile methods we keep each iteration a fixed length, this does not mean the schedule never slips. I recommend planning with a *buffer* iteration at the end that starts with no stories allocated to it. This gives management the time to take action by moving incomplete work to the buffer. But when the buffer overflows, you must add another iteration and own up to a schedule slip. This addresses the customer concern of not getting all their requirements.

Myth 4: Agile Methods Are Only for Programmers

Agile methods do help programmers, but the benefits extend far beyond code. At the 2004 Systems and Software Technology Conference (SSTC), the U.S. Government's Top 5 Quality Software Projects for 2003 were awarded. At this presentation, Linda Crabtree, a Development Group lead on the Patriot Excalibur Project (one of the award winners), talked about how her project was having trouble meeting schedules and keeping the customer satisfied [6]. After her team adopted XP, they began to hit their schedules more consistently, and customer satisfaction increased.

To understand how an agile method can have such a dramatic effect requires a deeper understanding of the first agile value. Traditionally, we plan new projects based on similar past projects. Schedules are often developed assuming personnel with skills similar to those on past projects will be assigned.

> *"With agile, instead of predicting schedule performance based on results from a different project with different people, we plan our team velocity continuously based on the actual team's current performance."*

Agile is different. With agile, instead of *predicting* schedule performance based on results from a different project with different people, we plan our team velocity continuously based on the actual team's current performance. With agile methods, this is possible because of the short iterations that provide actual team results early and often. Keeping each iteration a fixed length, referred to as time-boxing, is key to accurate velocity measurement.

Patriot Excalibur is not the only project reporting positive project management results using agile methods. I and other conference attendees heard a similar report at the 2004 SSTC from David Webb, a technical program manager for the Software Division of Hill Air Force Base [7].

Through agile methods, we are learning that people are not commodities. That is, you cannot just pull one *individual* off a project and plug another in and expect to get the same results. This fact does not change as projects increase in complexity or size.

It is important to note here that key to both of these reported successes was not only early visibility of actual team velocity, but also a customer willing to collaborate, as was reported by both Crabtree and Webb.

Myth 5: You Get No Design With Agile Methods

I heard a manager in a large company say, "We tried Scrum [8], but I wouldn't recommend it because we ended up with no design." Some mistakenly believe there is not time to get design done when using agile methods.

What is different with agile is *when* the design is done. With agile, we do not limit design to a fixed time slot within a fixed phase. In fact, we encourage deferring design details – not skipping them. What is often missed by those comparing agile to traditional methods is the extensive cost of design rework that frequently occurs during integration with traditional methods.

Agile encourages doing design at the optimum time (e.g. when data is available and high-level requirements have been clarified) to minimize rework and thereby maximize overall team velocity.

When you hear someone say, "We ended up with no design," what they often mean is that they ended up with no documentation of the design. This leads to the next myth.

Myth 6: Agile Methods Do Not Allow Documentation

Note that the second Agile Manifesto value is a relative statement. It is a myth that agile methods do not allow documentation. However, most agile methods are silent on this subject, leaving documentation decisions up to the project [4]. This includes both deliverable documentation and *process* documentation (e.g., action items, meeting minutes). This subject has also been referred to as the *ceremony* of the project [9].

When making documentation decisions, recognize that it is not a matter of being agile or not agile. There exists varying levels of agility, but when planning your project's ceremony, be aware that short iterations and high ceremony may place your project schedule at high risk.

Now let us turn our attention to extending agile methods to a broader

range of projects, including physically distributed efforts.

Extending Agile to Large and Distributed Projects

After working on three failed multi-organization physically distributed projects in the late 1990s, I spent a year researching similar projects looking for causes and solutions [10]. At the heart of the difficulties, I found communication breakdown leading directly to increased project integration risk. A colleague brought to my attention that the solutions I was advocating had similarities to the agile movement. Ironically, many of the agile experts were saying do not try these practices on large and distributed efforts.

If you attempt agile methods *out of the box* on large and distributed projects, you are likely to fail. This is because these methods require extensions to work on more complex projects. As I discuss these extensions, I will also explain some of the wrong ways to extend agile methods.

For example, recognizing the need to address the integration risk some distributed projects in the past have employed *heavyweight* architectures (e.g., formal front-end design reviews, extensive presentations, lengthy written documents). These same teams often failed to deliver acceptable working software. You can fail by collaborating too much, too little, or in the wrong places.

Recommended Extension 1: Agile Architecture

I recommend employing *agile architecture*, which involves first setting up an *agile architecture team*. Think of agile architecture like agile requirements – there are two levels. At the first level, a *small* strategically selected team rapidly develops a high-level agile architecture and documents the results. The goal of the agile architecture is to address the complete project scope and provide a minimum set of architecture compliance rules (e.g., hardware platform requirements, minimum interfacing rules).

The resultant product of the first level is a *thin* architecture document that includes a simple, high-level (but complete) diagram showing the major system components. This document also includes high-level assumptions and a brief description of each component.

Agile architecture is similar to User Stories in that it represents a commitment to talking to solve detailed architecture issues collaboratively with the agile teams during each iteration. Key to keeping the architecture agile is communicating the simple high-level diagram and the minimum compliance rules to each agile team. At the second level, the agile architecture team focuses on the high-risk areas for each iteration. Cockburn has referred to these areas as the big rocks [11].

Each specific architecture solution is documented by the agile architecture team through a *lightweight position paper*. These position papers may be maintained separately, or appended to the thin architecture document. The architecture grows over time as the solutions accumulate. Crucial to agile architecture scalability is the strategically selected architects who use their experience to determine where the architecture is best kept simple, and where the big rocks lie.

The use of agile architecture has been proven to be successful on past large distributed projects that have employed hybrid agile methods [12]. XP refers to *metaphor* [13] to address architecture, but metaphor is too weak for many complex efforts, especially when the team members do not reside at the same location.

Scaling Up Agile Teams: The Wrong Way

While communication breakdown plagues many large distributed efforts, improved team communication rests at the core of agile team success. Key to this improved team communication is the self-directed daily stand-up meeting.

I heard a manager on a large project that was attempting to scale up an agile method say, "We cannot afford a manager for every six to eight people." His project was holding daily stand-up meetings with more than 30 people. Unfortunately, when you scale up daily stand-up meetings this way the meetings tend to lose their self-directed quality.

Scaling up agile teams and maintaining improved team communication can be tricky. I recommend keeping the agile team's stand-up meetings to a reasonable size (less than 10), even if the full project has 500 or more people. Critical to the success of the daily stand-up meeting is the individual. Each must be heard. When stand-up meetings get too large, leaders start directing rather than listening and agile benefits are lost.

Some misunderstand the role of the agile team lead (e.g., ScrumMaster, XP Coach/Tracker) [8, 13]. A primary responsibility of the lead is to listen and then do everything possible to remove obstacles that are hindering the team. Too often this critical role is understaffed, especially when organizations attempt to scale up agile methods inappropriately. When the lead is not available to work on issues daily, communication breaks down and the team loses velocity – along with the key benefits of agile methods.

Another pitfall that has been observed when scaling up agile methods on large projects is a *stovepipe-mentality* among the individual agile teams. In other words, when a large project is partitioned into many small agile teams, improved communication inside each team may occur as expected, but at the expense of reduced communication across the full project.

Recommended Extension 2: Use Super Leads to Scale Up Agile the Right Way

One way to scale up agile methods and avoid the pitfalls discussed is to use *super leads*. Each super lead may oversee between three and five agile teams. It is important to understand that the super lead is not the agile team lead. Super leads may or may not attend daily stand-up meetings but if they do, they do not speak. The super lead's role is primarily a mentoring role for less experienced agile team leads, and a communication role.

Super leads review agile team plans and metrics (e.g., velocity charts) providing feedback directly to the agile team lead – not the team members. This approach can help address the observed lack of self-directed team skills on many large and distributed efforts. It is important that the super leads do not direct the agile teams; otherwise, we risk losing a primary benefit of agile methods – improved visibility of real status early and often.

The super lead oversees multiple agile teams with an eye on cross-team communication, ensuring the agile architecture team is engaged at the right time. This can reduce the stovepipe mentality.

I heard one agile team lead say, "The short iterations are causing my team to get burned out." This comment might be a warning sign to a super lead that help is needed. A key to successful implementation of agile methods is a self-directed team that can measure its own velocity, project its future velocity, and communicate the results up the chain. If an agile team finds itself working 80-hour weeks, this is a sign that they may not be accurately measuring their velocity, or they are not controlling their work tasks, or they are not communicating effectively the real status up the chain. A more experienced agile team lead will be able to spot these signs and help get the team back on course.

Even if they are in short supply, using your best leaders across multiple teams

can help facilitate communication on large and distributed projects, as well as help jumpstart agility on a broader scale in your organization.

It is worth noting that the super leads may meet periodically as a team. A similar concept (Scrum of Scrums) has been discussed by Ken Schwaber on scaling of agile methods [8].

Recommended Extension 3: Use Super Leads as Customer Proxies and to Aid Customer Communication

Often, especially on larger projects, customers cannot be on-site full-time. A primary motivator for having an on-site customer is to answer questions quickly so the agile team does not lose velocity. Sometimes customer proxies (e.g., subject matter expert) can serve this purpose, especially on large and distributed projects. When super leads are employed who also have domain experience, they can sometimes fill this role. Even if the super lead does not know the answer they might know who to call. The potential value can extend beyond just getting an answer to the immediate question.

I heard an agile team lead say, "My customer's travel budget was cut, so I haven't talked to him in a month." Just because your customer cannot be on-site does not mean you cannot build a strong relationship. As an example, if your customer cannot be at your daily stand-up meeting, consider institutionalizing a periodic phone call.

A key to agile team success is close collaboration with the customer. As projects scale up and people get busy it is easy to stop talking to the customer. With today's virtual communication options and what we know about the importance of customer collaboration for success, physical location and project size are no excuse for lack of communication [10]. It is worth noting here that most agile methods do not require an on-site customer. It is, however, a required practice of XP [13].

Recommended Extension 4: Lightweight Guides and Enablers

Some believe that with agile methods the written word becomes less important. In fact, what we have found is that when we extend agility to large and distributed efforts the written word takes on increased importance [10].

This is largely because we cannot get to every individual face-to-face on large projects every day. But this does not mean these projects cannot still gain the benefits of agility. However, to do so requires that personnel be trained in what needs to be written, how to write it from an agile perspective, and – just as important – what is best left unwritten and handled through less formal means.

I recommend that organizations develop lightweight process enablers to help guide agile teams. It is worth noting that I do not recommend tailoring down *heavyweight* processes in support of agile methods. This has been shown to be fraught with difficulties. I also recommend that organizations institute leadership-at-a-distance training for those who must collaborate with team members and customers who cannot always be physically present.

Conclusion

Some have asked, "How can you be agile and collaborate?" Just watch an agile team in action in a daily stand-up meeting and you will see the right level of collaboration focused on the right things, without wasting time on unimportant matters.

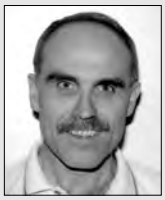
Agile is not about customers living without all requirements; it is about breaking through to the grassroots level, making real status visible and acted upon sooner, which ultimately provides greater value to the customer.

Organizations of all sizes are today taking a serious look at agility. If it is not happening in your organization yet, you might be missing the next big velocity boom!◆

References

1. Cockburn, Alistair. Agile Software Development. Addison-Wesley, 2002: 215-218.
2. Cohn, Mike. User Stories Applied. Addison-Wesley, 2004.
3. Larman, Craig. Agile and Iterative Development: A Manager's Guide. Addison-Wesley Professional, Aug. 2003.
4. Cockburn, Alistair. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, 2004.
5. Sutherland, Jeff. "Continuous Scrum." Scrum Study Group Registry. 9 Feb. 2005 <http://wiki.scrums.org/index.cgi?ContinuousScrum>.
6. Crabtree, Linda. "U.S. Government's Top 5 Quality Software Projects for 2003." 2004 Systems and Software Technology Conference, Salt Lake City, UT, 21 Apr. 2004 <www.stsc.hill.af.mil/crostalk/2004/07/0407Top5_PEX.html>.
7. Webb, David. "Combining Discipline and Agility: Using Agile Techniques to Enhance the Team Software Process." 2004 Systems and Software Technology Conference, Salt Lake City, UT, 20 Apr. 2004 <www.stc-online.org/sstc2004proc/cfmfiles/PresentInfoEntry.cfm?abid=251>.
8. Schwaber, Ken. Agile Project Management with Scrum. Microsoft Press, 2004.
9. Kroll, Per. The Rational Unified Process Made Easy. Addison-Wesley, 2003.
10. McMahon, Paul. E. Virtual Project Management: Software Solutions for Today and the Future. St. Lucie Press, 2001.
11. Cockburn, Alistair. "Extending an Architecture As It Earns Business Value." Technical Report TR 2004. Salt Lake City, UT: Humans and Technology, 14 Jan. 2004 <http://alistair.cockburn.us/crystal/articles/eaaaiebv/extendinganarchitecture.htm>.
12. Procuniar, Don, Paul McMahon, and Dennis Rushing. "AVCATT-A: A Case Study of a Successful Collaborative Development Project." Interservice/Industry Training, Simulation and Education Conference, Orlando, FL, 26-29 Nov. 2001.
13. Beck, Kent. eXtreme Programming Explained: Embrace Change. Addison-Wesley, 2000.

About the Author



Paul E. McMahon is principal of PEM Systems, which helps large and small organizations as they move toward increased agility. He has taught software engineering at Binghamton University and conducted workshops on engineering process and management. McMahon is author of more than 25 articles, including two on agile development in the October 2002 and May 2004 issues of CROSSTALK, and author of "Virtual Project Management: Software Solutions for Today and the Future." McMahon is a frequent speaker at industry conferences, including the Systems and Software Technology Conference.

PEM Systems
118 Matthews ST
Binghamton, NY 13905
Phone: (607) 798-7740
E-mail: pemcmahon@acm.org