Applying Agile Principles in the DoD: Seventh Principle

featuring Mary Ann Lapham and Suzanne Miller

Suzanne Miller: Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center in Pittsburgh, Pennsylvania. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

My name is Suzanne Miller. Today, I am pleased to introduce you to myself and Mary Ann Lapham. Welcome to our ongoing series exploring Agile principles and their application across the Department of Defense [DoD]. In today's installment, we explore the seventh Agile principle: *working software is the primary measure of progress*. But first, a little bit about Mary Ann and me.

Mary Ann's work focuses on supporting and improving the acquisition of software-intensive systems. For Department of Defense programs, this means assisting and advising on software issues at the system or segment level. In addition, she leads research into software topics that are germane to acquisition of software-intensive systems, like Agile methods.

My research focuses on synthesizing effective technology transition and management practices from research and industry into effective techniques for use of Agile and lean methods in regulated settings like the DoD.

This is our seventh podcast exploring the real-world application of Agile principles in Department of Defense settings. If you like what you hear today, there are already six podcasts in the series, and we will eventually have 12 in all, one for each Agile principle behind the Agile Manifesto. As a reminder, the four values and 12 principles of the Agile Manifesto can be found at www.agilemanifesto.org/. Today, Mary Ann, let's talk about that seventh principle: *working software is the primary measure of progress*. Take it away.

Mary Ann Lapham: I don't know where to begin, mainly because in a typical DoD project, you don't get working software [software] until almost the end. I mean not working software that somebody can take out and take for a spin.

This is so different, and yet it's a wonderful thing.

Suzanne It is different from Big Bang. But, let's be fair, there are incremental design processes that aren't Agile that can produce software.

Mary Ann: That's true. That's true.

Suzanne: But, it's that primary thing.

Mary Ann: It's the primary thing that, *Do you mean I can actually have some software in three-to-six months that I can try out? And, I don't wait to the very end to integrate everything together, so I find out early on what my integration risks are and some of my other testing risks and other things that I forgot?*

Because, as we all know, even though in a traditional environment all the requirements are defined up front, there are always one or two or three or five things…

Suzanne: Or more.

Mary Ann: Or more, yes, that we forget and we add in. The later you add them in, the more problems it's going to cause either for integration or changing of a design, or architecture, or all those things that cost money. To have actual working software where you can give it to your end user and let them take it out for a spin and see if it really works. And, then find out, *Well, yes I like that, but can we do this different in the next piece that you're going to build and that different over there? And, oh, we need to add this.*

Suzanne: Or this technology has changed, and you can't rely on it being there anymore.

Mary Ann: Right.

Suzanne: All these kinds of things play into this.

Mary Ann: Yes.

Suzanne: Now the thing that was striking to me, the first time I read this way back in the day, and thinking about the projects that I had worked on, was the aspect of how I get measured. When you're a software programmer, when you're a coder or even as a designer, you don't get measured on the working software until the very end. How many defects? All that kind of stuff.

But, there's a huge amount of time where you are working on design documents, working on requirements refinement, working on interface descriptions, working on everything, working on essentially everything but the software itself. So, this mindset that working software essentially takes precedence over some of the other artifacts that we are accustomed to, this is a very big shift for our DoD audience.

Mary Ann: It is a big shift. The other thing that this makes me think of when you start talking about measuring, most of the very large systems by mandate are required to do earned value management. It is a fairly rigid system where everything's defined and you don't change things. But, if you're going to do it on working software…

Suzanne: …and, if you're allowed to change requirements at the lower level.

Mary Ann: …and, you're allowed to change things, then how does that go? That has all kinds of implications on how you do that. There are a lot of people out there working on that particular problem, because it is an issue. It depends at what level you measure. If you measure at a high enough level, like at the release level not at the sprint level, where it releases multiple sprints…

Suzanne: …that actually cohere.

Mary Ann: Yes.

Suzanne: I want to be clear about something to our audience. We are not naively saying that you're going to get an entire software system…

Mary Ann: Oh, heavens, no.

Suzanne: …in three months or even six months. But, the whole Agile design philosophy is to look at—instead of looking horizontally across a layer from an architectural viewpoint—you try to look vertically. You try to say, *Well, this thread, if I can do this piece, if I can tell you how this service is going to work…*

Mary Ann: Right, a complete feature.

Suzanne: *…then you can give me feedback instead of me giving you the whole user interface or giving you the whole database.* So, that's a very different way of implementing.

Mary Ann: Right, it is. You're going vertically as opposed to horizontally. So, vertically you cut across all the different pieces and parts that would make up a feature, and you see if they all, one, play together; and two, give you the kind of results you're expecting as opposed to, as you said, put in all the user interfaces and then add the database and then the other pieces and then find out, *Oh, that doesn't work so well*. But, if you take one feature if you will or mission thread…

Suzanne: Yes, mission thread is a nice way to do this.

Mary Ann: …and, see how it works across all the different pieces and have an actual working small piece, and then you add to it. So, you have to figure out, *How am I going to do this. How do I chunk it up so I have a foundation?*

Suzanne: So this has architectural implications.

Mary Ann: Yes.

Suzanne: If I'm going to go this way and I'm going to try and focus on a working software early, I have to have enough architecture so that I've got confidence that what I'm doing is actually going to play with the other pieces. Software doesn't exist in a vacuum. It exists with other elements of the system. *And*, I've also got to make sure that I communicate with my architecture folks so that as they're building out the architecture, if I find out things at implementation that don't work, I can't connect this service to that service with the kind of interface that they have said we should use, for example. I've got to have a path for communicating with them.

So, this goes into some of the cross-functional-team kinds of stuff. This has implications across a lot of aspects. To me, it's one of the defining elements of what is different. People ask us all the time, *What's different about the incremental design and incremental build, the way that it's defined in some of the DoD Guides?* Like 5000.02 does have guidance on incremental.

Suzanne: But, it doesn't focus on this. It's not iterative, and it doesn't focus on this primacy of working software as the way that you're trying to move things forward.

Mary Ann: If you stop and think about working software, well, if it's just a small program, it's probably not a big deal when you get into some of the medium- to larger-size programs where there's more than 1 team, because there has to be. And, a lot of people are trying to use Agile. And, there's some methods out there like SAFe [Scaled Agile Framework] and DSDM [Dynamic Systems Development Method] which can be used.

Suzanne: And DAD, Disciplined Agile Delivery.

Mary Ann: And Discipline Agile and all those things. They are incorporating architectural runways, making sure the architecture is there when you need it. Working software isn't necessarily working for…my question is, *Working for whom?* What you want to build first usually is the infrastructure so other people can plug into it.

Suzanne: It might be working for other teams, sometimes.

Mary Ann: Could be working for other team. You're building the infrastructure so the other teams can build upon it. So, it's working structure for somebody that can use it, but it may not be the end product yet. I've seen that several times.

Suzanne: It's desirable to get the end user in there, but you can't always get that right away.

Mary Ann: Yes, well, not until you get some kind of infrastructure, especially for a larger program, in place.

Suzanne: A couple of examples: We had one example that Mary Ann and I both worked on where the team was trying to do this. They were doing exactly what she said: They were building the infrastructure first, based on the architecture, and so that everybody would have a chance on the team to work against it and look at how their requirements played, and all that kind of stuff. We were there to help them get ready for a preliminary design review [PDR], which is traditionally a review of documents. But, this team had a mix of infrastructure elements that were already built. And documents. As is typical in Agile, they [the documents] were not as refined in terms of their detail because you are trying to get feedback.

Mary Ann: Well, some were. Some were. It was interesting there. Not only did you have some working software, but you had documents that were ready for PDR, like you normally expect. There were some documents that would be what you would expect at a normal CDR [Critical Design Review].

Suzanne: Right, Critical Design Review.

Mary Ann: They were more detailed. Then you'd have some that nobody had even worked on yet because they were so far down the road of what sprint they were going to be in.

Suzanne: Or even what release.

Mary Ann: Yes, or what release. We're not going to get to those until next month or next year even, depending. So, it was fascinating. Then try to get your user committee that typically comes to these milestone reviews to understand this.

Suzanne: There were some roles that are accustomed to seeing the entire software spec, all in great levels of detail, that were very upset, quite frankly, that they didn't have everything they were accustomed to looking at. Some of them were excited about the fact that, yeah, but, *wow, we've got this piece of the infrastructure. We can do an even better job of reviewing because we can see where they're really going*. There were others that wanted to have nothing to do with the working software. All they wanted to look at was the documents.

This is a case where the cultural aspects of Agile are sometimes ahead of where some of the stakeholders are in the acquisition process. This working software principle really brings that to the forefront when you see that.

Mary Ann: Very fast. Very fast. And, it's fascinating to watch the different reactions. Some people are like, *Wow, yes, this is really cool.* Then you have the others who sit back and go, well, they're not going to get my pass on this. So, depending on who gets the final say, you either have a very successful milestone review in an Agile environment or you spend lots and lots of money to get all the documentation that you would expect in a traditional environment, adding costs to what you thought was going to be something different.

Suzanne: We have another Agile program that we've worked with that takes this very much to heart. They have user involvement: Every six months they have a program management review where the demonstrate everything that's in the works, anybody that is providing them funding. They have several major commands that provide their funding, they are a military system.

They all come together and they get to see everything and provide their feedback. Based on how much money they've spent, they get bucks, system bucks, that are things they can spend to get changes made. It's quite interesting to watch that negotiation go on. It is all based on, *We're looking at something real*. That, to me, is one of the biggest attractions for end users and acquirers that are working in this space, is, what we call in the Agile community, *I get to fail fast*. If we're going in the wrong direction, we know it really fast because people see something and say, *Wait, wait, wait, wait, wait, can't do that and here's why*. I have seen cases where things have come out in these kinds of reviews that never would have come out in a documentation review or technical interchange meeting because you didn't have the level of insight that using the software gives you as opposed to just looking at people writing about how the software's going to work.

Mary Ann: Right. In those instances, you've saved yourself time, money, and lots of heartache. Normally, you wouldn't see those kinds of issues until the big bang integration towards the end. By then you're so far down a path of, *Oops*. So, then you have to fix it, and it's going to take time and money and patience.

Suzanne: You may not be able to fix everything in time.

Mary Ann: That's true.

Suzanne: And, you may have certifications issues and all those other kinds of things that come down.

Mary Ann: Right.

Suzanne: So, let's talk a minute about how do you make this a successful practice in your acquisition. How do you make working software the primary measure of progress?

Mary Ann: My immediate answer is you have to plan for it. Contrary to what a lot of people think, *Oh, Agile, that's just that crazy unplanned for, cowboy…*

Mary Ann/Suzanne: Cowboy.

Mary Ann: Yes.

Suzanne: We both said that at the same time.

Mary Ann: How long have we heard that? *You are cowboys*. My immediate response to cowboy is, *Yes, that's interesting*. However, if you are consistently delivering working software, meaning tested, ready to go out in the field. Now, minus the C&A, certification and accreditation. That is an issue.

Suzanne: Whole different thing.

Mary Ann: Different thing. But, if you just have working software that's been tested and is verified, ready to go, how can you do that consistently every two-to-four weeks if you don't have a plan, if you aren't structured and disciplined?

Suzanne: That's right, and transparent and visible and all the other principles we've been talking about.

Mary Ann: You can't not plan and be Agile. A lot of people say, *Well, I'm Agile*. Until you really look at what they're doing. They're not.

Suzanne: I always go back to the Eisenhower quote about how the plan is not what's important, it's the planning.

That is, I think, the difference that we see in good Agile teams: they take the planning very seriously. They plan every two weeks or however long their iterations are. The point is that it's a different kind of planning. It's not the "Big Bang", I'm going to plan every detail six years before I am going to actually execute that, which is an impossible situation to put people in. The planning is iterative as well as the execution.

Mary Ann: It's planning for the next two weeks. Now, even in normal ordinary circumstances, can you really plan what you're going to be doing five years from now, in gory detail, and it never changes?

Suzanne: Not in my life.

Mary Ann: But you could probably plan what's going to happen in the next two weeks.

Suzanne: Closer.

Mary Ann: A lot closer, depending.

Suzanne: Depending on what else is going on. Actually that's really the reality of it.

We set the boundaries of an iteration because we say, *Well, we think the next two weeks it's safe, right? For us to be able to say we're not going to allow any changes over the next two weeks*. You get up to a month, and now, somebody is going to be getting on my case to do something different quite frequently.

We've talked at other times about iteration length. That is one of the reasons for that two weeks. It is short enough that it's likely that you're going to not get bothered by new things in that time frame, but it's long enough to actually get something done if you're working intensely, which most teams are.

Mary Ann: Then, to not be bothered by other outside influences is a really difficult thing in most environments today.

Suzanne: We've explored this in a little bit of detail, although this is one of the ones that I think we could probably do six or seven podcasts just on this one [principle]. We'll go ahead and stop it here. I want to thank you, Mary Ann, as always, for joining us today.

Mary Ann: Very welcome.

Suzanne: In the next episode, we'll be exploring the eighth Agile principle. It relates to something that I was just saying; Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. So, we'll get more into how do you craft your release times and your iteration times, so that you can have sustainable development, because that's a very important aspect of this.

Listings for papers, blog posts, and podcasts related to SEI research on Agile Adoption in DoD, can be found at sei.cmu.edu/acquisition/research.

If you'd like more information about the SEI's recent publications in all areas of our work, you can download all of our technical reports and notes at resources.sei.cmu.edu/library/.

This podcast is available on the SEI website at sei.cmu.edu/podcasts and on Carnegie Mellon University's iTunes U site. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you for listening.