Agile in the DoD: Ninth 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 sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University in Pittsburgh, PA. 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 to you to myself and Mary Ann Lapham. Welcome to our ongoing series exploring Agile principles and their application across the Department of Defense. In today's installment, we explore the ninth Agile principle: *continuous attention to technical excellence and good design enhances Agile*.

First, a little bit about myself and Mary Ann. 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 and/or segment level. In addition, she leads research into software topics that are germane to acquisition of software-intensive systems including the SEI's research in adoption of Agile and Lean methods in regulated settings like the DoD.

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. This is our ninth podcast exploring the real-world application of Agile principles in Department of Defense settings. If you like what you hear today, there are already eight podcasts in this series. We will eventually have 12, 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 technical excellence and good design; things that the SEI is very much behind. What are some of the challenges, and what have we seen in Agile settings related to technical excellence and good design?

Mary Ann: Well, there are several areas that we probably should explore. There are things like architecture runways, having a good design—not a complete design but a good design—up front contrary to what many people think.

Suzanne: Some people call it *enough design up front*.

Mary Ann: Yes. Enough design up front. Not no design, because you do want to have…

Suzanne: …and not big design.

Mary Ann: …and not big design because there will be some emergent design, and you have that even with the big design. You find out, *Oh, that didn't work so we really need to go this way*. This allows that to happen without changing too many things.

Then there are architecture runways, some of the multi-team approaches like SAFe [Scaled Agile Framework] and so forth use that, having just enough architecture but continuously improving on that. Again, technical debt is what you start with when you start looking at these programs. *What is broken? What don't you catch within the actual sprint?*

Suzanne: *What do you push forward that you don't understand how to deal with?*

Mary Ann: *How much of your sprint, each sprint, even in new development, do you allow to fix bugs that you've found as you are doing integration and so forth? What does done mean? Does done mean there are absolutely no more defects in my current sprint?* Probably, but what happens when you go to the next sprint and add more to it and then you find an unfound bug from before? Again, technical debt. How do you deal with that?

Suzanne: I want to talk about that for a second, technical excellence. We talk about Agile allowing good teams to really have superior performance. That is one of the basic aspects of Agile; there is an assumption that the people on the team are competent at what they do. So, you have got to have that competency for coding. You have got to have some competency for design, for detailed design. You have got to have some competency in unit testing. You have got to have some competency in integration. So, there are assumptions about what kinds of things your cross-functional team is capable of, and those are things that if you have them, it will enhance what you are able to do. If you don't have them, you are going to build technical debt. You are going to build in defects. Then, what you are going to find is instead of a very small portion of each iteration being dedicated to dealing with defects that were introduced in a prior iteration, you are going to have a lot more of your work focused on catching up with defects that are already in place.

We have seen that in some programs where the teams were either new, were brought in late in the game and were added in, didn't really have a good understanding of the design. You can see

just from looking at the measures, their technical debt is higher, their defect rework time is higher. All those things are because you weren't able to apply…you weren't able to get the technical excellence the first time through. That is one the things that people often don't think about is that you have got to have those competencies in the team to be able to achieve that technical excellence.

Mary Ann: It gets into the question that a lot of people say, *Well, a good Agile team has to be very highly skilled, your A game, if you will. Your A players. An average kind of guy really won't play well in an Agile team*. Well, that's not true. People will rise to what you expect them to do. Now, granted it's nice to have an A team player on your team because everybody else wants to emulate them and be like them and that always helps. But, if you expect a lot out of a team, they will rise to the level you are expecting.

Suzanne: They will, but they have also got to have the help to get there.

Mary Ann: They have to have the help to get there. One of the things we haven't touched on either is as you are going through the Agile process, at the end of every iteration, there should be some kind of demonstration. Now, that shouldn't be just a, *Check the box. Yes, we had a meeting*. It should be actually show me that what you have works. Then, if the demonstration is successful…

Suzanne: That's part of the technical excellence too.

Mary Ann: That is part of the technical excellence. That is a checkpoint that you can look at and say, *Yes, it really does work* or *We've got problems*. Then you have to make the decision do we go forward.

Suzanne: Do we correct what we need to correct?

Mary Ann: Do we correct? Because if you don't correct, in some cases you may want to not correct, but then you have to do a balance between, *How much technical debt are we creating?* And, *What is the implication of that downstream*? And, do it consciously. Don't just let it roll along.

Suzanne: Let's talk for a minute about architecture and design in the Agile context. One of the principles…if you talk to any of the Agilists, Scott Ambler is a big proponent of Agile architecture, Dean Leffingwell included explicit architectural stories in the SAFe method. He and others have created…

Mary Ann: Well, he has the architecture runway in his SAFE method.

Suzanne: Those are all explicit elements of their above-the-team-level process. What that implies at the team level is that you are always paying attention to design, not just to implementation. That is a role that you have to get people to understand, that they are not just coders on the team. They are coders. They are requirements interpreters through the stories, conversations they have with their product owners. They are designers. They are coders. They are testers. There may be specialty people that help them with documentation and UI [user Interface], but you are actually asking your programmers who would be traditionally just doing code to actually be more of an engineer. And, that's a shift for some people.

Mary Ann: Well, it is a shift, and it is what one of our collaborators in the Agile collaboration group we have called a T-shaped man. Very deep in some expertise, some area, maybe coding, maybe testing, maybe design.

Suzanne: UI.

Mary Ann: UI, whatever, but broad. So, they understand all these other disciplines and how they play together. They may not be an expert in them, but they certainly understand that, *Oh, this is a UI problem, and I have to go get my expert because it's above my depth*. But, there is their area where they will be asked.

Suzanne: Databases.

Mary Ann: Databases. If you are the database expert, but broad in other things, if there's a database issue they'll come to you and say, *So, from your database background, what's the answer to this one?* So, it's a different kind of feeling. It is almost like a Renaissance man in software engineering.

Many people don't naturally generalize—because we have over the years become very specialized, *I'm a coder, I'm a designer, I'm an analyst, I'm a this, I'm a that*—well, now it's back to you have to be a generalist across all areas but be a specialist in one and use those skills and get the right set of people that have specialties in all the areas you need but still understand the other disciplines, so that you can work together as a cross-functional team.

Suzanne: There are some implications for this when you talk about the acquisition side. We are seeing a lot of different kinds of contracts where you have got a type of contract that allows you to have a pool of resources in terms of contractors, and then, each task order, you select somebody to do that task order. One of the implications for acquisition personnel is you have got to take responsibility for good design because if you are going to be asking all these different people to come in and work on your program, you have got to know what pieces will work together, what pieces you are having problems with, how your architecture is evolving. So, there

is more responsibility, in my mind, on the acquisition side for understanding your architecture and understanding your design, if you are going to use some of these contracting mechanisms.

There is one that is typically called IDIQ, indefinite delivery, indefinite quantity. That is a popular method for Agile developments. One of the things that is very prevalent in Agile is trying to create architectures that allow for components to be developed independently. You could only do that if you know that the architecture actually supports that. So people that say, *You don't have to do architecture if you are doing Agile*, or *You don't have to do design up front if you are doing Agile*, we have seen over and over again that if you ignore the design, if you ignore the architecture, you are going to not be able to have independent components, and you are not going to be able to have a system that meets the user's needs.

Mary Ann: Well, it all gets back to planning. A lot of people think, *Well, Agile is just this ad hoc thing that is cowboy programming, cowboy whatever*. You have to plan. If you are going to do Agile and have those teams working consistently and having attention to detail and attention to design and attention to quality. Quality is built in. It is not added on at the end. Everybody has to be aware of that. They have to be the cross-functional team that they are. They have to make sure they understand where they are headed. If you don't have those forward-thinking things, you are toast, basically.

One of the things we saw on many of the programs is what they used to call sprint zero. I haven't seen that term in a couple of years, but it was the planning piece before you even got your teams together. They had what they called a sprint zero where they come up with the initial architecture. *What's my sandbox?* In some cases, they would use that sandbox to define their security environment. And, *What are the things I need to have on my individual teams?*

Suzanne: What platforms?

Mary Ann: What platform. They have a lot of those planning decisions done up front. Not everything, not in the gory details that you would see in some of the more traditional programs, but enough so the team could start. They knew what their boundaries were. Then, they were allowed to do what they call emergent design and emergent properties, because they know…

Suzanne: Within the framework of that platform architecture.

Mary Ann: Within the framework they had—and they would early on define—whether or not that framework worked well or not, and they would have to add to it.

Suzanne: We know one program that has been evolving. They are in a sustainment environment, and they have been evolving over 10 years. And, they can tell when it is time to do a redesign. They can see by how the technical debt is starting to emerge and analyze their code base to

understand when they need to do a major rework of a design. They take out some time, they take more than one sprint, sometimes a whole release, to do a rearchitecting, putting it on a new…They had a case where the DoD was moving to service-oriented architectures [SOA] and they were on a traditional client server. So, they took time out from their normal schedule. They made sure that their users understood what they were having to do. But they were able, in pretty short order, to rearchitect this thing because they had a very good understanding of what the prior architecture was, what design decisions had been made, and they understood how to get to the next step. That is pretty powerful because I know other programs that have struggled mightily with that shift over from one type of architecture to another.

Mary Ann: Updating the technology is one of the big problems.

Suzanne: Technology refresh is a big deal.

Mary Ann: Programs that are going from client server—we will use that example—to SOA, to whatever, that is a whole different cloud game. The cloud is back in vogue now. So, how do you do that? I have seen programs flounder because they can't get into the new technology because they just don't have enough understanding.

Suzanne: One of the advantages of Agile in that setting is you can do some small things to try things out through some of these innovation sprints, through a platform, replatforming kind-of-a-sprint, and get some of that information and do it while interacting with your users and your customers because that is the thing they didn't give up. Even when they were doing what would be considered internal architecture stuff, they still created demonstrable prototypes to say, *If we did it this way, how would that work? If we did it that way, how would that work?*

Mary Ann: Which was very important because some of their users had architectures that they had to work within on their particular constraints that they had to know that certain architectures that the program wanted to go to just wouldn't work when they tried to roll them out to their user's environment. So, they had to take that into consideration. They found that out early because they were working with their users and saying, *OK, we are thinking of going here. What does that do to you guys?* The users knew what their networking was and what worked and what couldn't work, and they fed that back. That kept the program from going down the wrong direction because they did these early exploratory sprints.

Suzanne: So, I want to summarize a couple of things. One is technical excellence is not just about being the best coder, not just about being the best tester. It is about having the team that has the right competencies for the problem that you are trying to attack. It is something that management has to pay attention to all the time in terms of making sure people have the skills, sometimes have the tooling environment, that they need to produce technically excellent work.

The other thing is good design crosses sprint boundaries, it crosses iteration boundaries, it crosses release boundaries. You have to understand what level of design and architecture are needed for you to proceed forward. You want enough design up front, not too much, but you want enough design up front, so that your teams can produce functionality, can get that interaction going early, so that they know that things are going to work right.

These are things that are very consistent with this Agile principle, and they are things that are achievable. We have seen this achieved very well within DoD constructs. And, we have also seen when it's ignored… Sometimes ignoring this principle alone will make it seem like Agile is failing. I think that is the other thing we want to say. When you see what appears to be an Agile failure, often it is because one or more of these principles have been violated and certainly in the early days of adoption of Agile in the DoD, this attention to design and technical excellence wasn't quite as emphasized as it has been later on. So, that is something to watch for when you see those kind of failures.

Mary Ann: It is. It is build quality in, and what is the value you are getting as you go along. Not value at the very end, but you get value at each at each iteration and then a release when you can deploy it. So, it is the value as you are going along, not all at one big bang at the end. And quality, always quality across everything.

Suzanne: Mary Ann, thanks for joining us today. In the next episode in this series, we will explore the tenth Agile principle, simplicity: the art of maximizing the amount of work not done is essential. We know that not much is simple in the DoD so that should be an interesting conversation.

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 would like more information about the SEI's recent publications and 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

Editor's Note: This transcript has been edited slightly to update links and improve readability.