Agile in the DoD: Eleventh 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. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

My name is Suzanne Miller, and 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. In today's installment we explore the 11th Agile principle, *the best architectures, requirements, and designs emerge from self-organizing teams*.

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 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 11th podcast exploring the real world application of Agile principles in Department of Defense settings. If you like what you hear today, there are already 10 podcasts in this series, and 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 architecture requirements, designs, and self-organizing teams.

Mary Ann: When you hear the principle, *best architectures, requirements and designs emerge from self-organizing teams*, any self-respecting DoD manager would run screaming from the room. *What do you mean emerge from a self-organizing team? Oh my gosh!* You have to understand what those terms mean. That is the key to understanding what *this* means. It doesn't mean chaos and people are just saying, *Go do something*. Heavens no. Self-organizing means you give the team boundaries.

Suzanne: Boundaries, right.

Mary Ann: And, say, *OK. Here is the problem*. You give them an initial skeleton, if you will, of an architecture. You don't just say, *Go make one up*. It is stuff that people would call maybe sprint zero, depending on who you talk to. Then, you let them go solve the problem.

Suzanne: Let's talk a little bit about the architecture thing because that, in the larger Agile community, has been a topic of debate for some years. Although I think we are seeing some resolution of that where most Agile teams in commercial settings are starting to acknowledge the role of architecture, not just emergent architecture but actually some design up front. They call it *just enough design up front* as opposed to *big design up front*. We are seeing frameworks like SAFe—the Scaled Agile Framework is one—that explicitly talk about things like architecture stories and architecture runways and doing things in a way that allow you to evolve the architecture at the same time that you are implementing the functionality.

Mary Ann: Yes, and SAFe requires that you have enough architecture available to keep ahead of your teams.

Suzanne: Sort of platform-centric.

Mary-Ann: So, platform centric. *So, I know I need to have in advance figured out what platforms I'm going to be using*. If there is some infrastructure that has to be built, those usually are the architecture stories that you were talking about. There is a team, maybe one or two Scrum teams—it could be just one—that puts together this infrastructure framework that other people can use. That might be the first delivery. Now, they will have potentially shippable code at the end of their sprints, but…

Suzanne: But, not for the end user. It is really more for the developers and testers.

Mary Ann: Not for the end user, it is deliverable to a different "whom". It is deliverable to the other developers and testers and people that are building the system.

Suzanne: And, other stakeholders like information assurance folks and people like that.

Mary Ann: So, if that is the first delivery it is to users but not the end users.

Suzanne: And, we have seen that.

Mary Ann: Yes, we have.

Suzanne: We have seen cases in the Air Force where they basically do this. The first couple of releases are actually releases to the development team and internal stakeholders where they are essentially providing the platform. So, they are actually working on that architecture. These are cases where they are living inside of an embedded system. They are living inside a larger systems engineering framework. So, they have to be able to deliver to that part of the team, and that is their customer in that sense.

Mary Ann: And, that is their customer. And, because they are living inside of a bigger entity if you will, they have to take their cues for what their architecture is going to look like from that entity. So, they don't do it in a vacuum, per se. They actually look at the system they are going to be working, living within, and use that as the guidelines.

They also have to use guidelines from their environment. One of the programs we know of, they bring in their security people very early and they build in, bake in basically, all the things they need for the controls to work in their environment, the applicable security controls. That becomes part of the infrastructure, and that becomes part of the boundaries that the team has to work within.

So, yes, they have self-organizing teams because you give the group a problem, *Go solve it*. But, you have given them the boundaries in which they have to play, and you have given them some tools to go with it. You just don't say, *Here is a blank sheet of paper. Go do it*.

Suzanne: Right. I think that is the thing that in the Agile community at the beginning, it was that blank sheet of paper. Let's be honest, when you look at the history, many of the systems that Agile methods were tried on, especially early methods like extreme programming, were on legacy systems where you already had an architecture and the functionality was being revised. You were adding functionality, but you really had an infrastructure to begin with. In that case you may not need as much emphasis on a pre-designed architecture.

Mary Ann: Or, the programs they were doing, the early stuff, was really small. Agile came about, came out of small, small developments, and now people are expanding it. Because the idea is, they figured out how to scale. . It's not just SAFe, there are others out there. Scott Ambler has one. DSDM, which is a form of Agile that is most popular in Europe, has one. Craig Larman has one. So, there are several out there. You need to go look at those if that is where you are headed.

Suzanne: Now, the self-organizing part of this is the other part that, as you said at the beginning, sends the program managers running screaming from the room.

Mary Ann: *I have no control. Who is going to make sure they're doing the right thing at the right time. Not telling them how to do their job but keeping some kind of order and movement in the right direction.*

Suzanne: This is really where the learning part happens. This is the crux of the learning. When I am on a self-organizing team, within the boundaries that I have been given, I have a lot of space in which to try out different things. I have a lot of space in which to work with other people, cross functionally, to get better solutions. So, this isn't just about the architecture, this is also about dealing with the requirements, understanding the requirements, dealing with the design elements that are specific to this functionality and using all the resources across the team to make sure we get the best solution. That is a real big win for places like the DoD that tend to be stove-piped when left to their own devices, where these teams are not just the developers. It is the developers, the testers, the user interface people, the IA [information assurance] people, and getting those people to work together in a self-organized manner is a challenge. But, it is also one of the great benefits that we have seen.

Mary Ann: Well, it is. One of the things you hear a lot about is developers do their stuff, and the analysis people create the requirements. The developers develop them. Then the testers get in and say, *I can't test that. That's not a testable requirement.* If they are involved up front…

Suzanne: They are all working together.

Mary Ann: They are all working together. Not only will you get good requirements that the developers understand but you get requirements that can be tested.

Suzanne: You get acceptance criteria.

Mary Ann: Right, you get acceptance criteria for each one.

Mary Ann: So, you are killing a lot of the issues you see in more big, upfront design and, you know, the big bang.

Suzanne: Yes, big design up front.

Mary Ann: Big design up front, I can never say that right. For the more traditional systems you are eliminating some of the issues you see in those, and you reap benefits because you don't have those issues further down the road where it's more costly to fix them.

Suzanne: Now, I will also say we have seen some teams that have had problems because they didn't have that cross-functional thing happening within the teams.

Mary Ann: That is true; they have mini waterfalls.

Suzanne: We know of at least one case where information assurance people did not want to be involved. They saw it as a loss of independence. They were not really appreciative of the fact that you are building architectures early. They wanted to see traditional documentation. So, that is something that can happen when you are dealing with these kinds of issues. You have got to look beyond the team to what's your environment and make sure that you get those folks enlisted. Or, you figure out how you're going to deal with them so that you actually meet your obligations to them because if you don't, then you are going to have a whole different set of issues.

Mary Ann: Yes, and, and the other group that falls in that same category is the operational test folks because they are, by regulation, supposed to be independent, so how do you get them to play in the space is another issue you need to start figuring out up front.

Suzanne: And, we have seen more with the developmental test people, the DT and E people. We've seen those folks be directly involved in the teams. It is less common for the operational [testers].

Mary Ann: Less for the ops. I know some of the groups are looking at it. It is a different way of doing business, not only from the way they traditionally want independence, but it is the way they staff their teams. So, there are those issues too that, if you decide to go this Agile route, sounds really cool. Well, there are a lot of issues you have to overcome, and you have to think about it. And, contrary to popular belief, you have to plan.

Suzanne: Now, our colleagues, Ipek Ozkaya and Rod Nord, who are working directly in Agile and architecting, they would beat us silly if we didn't say something about technical debt here.

Mary Ann: You are right

Suzanne: So, the concept of technical debt is that as you accumulate implementations, you sometimes, either through defects or through conscious decisions, say, *I am not going to deal with this right now*. So, you accumulate these areas where you have to later go back and fix stuff. In the Agile world they would call that re-factoring, but there is a cost to that. You can't just be moving forward. If you have got technical debt that is accumulated, you can't just move forward and ignore it. At some point, it is going to come back and bite you.

So, that is the other piece about the self-organizing teams is that the teams need to have enough power to understand what they are doing and when they are pushing things forward. But, there also needs to be some governance that helps them to understand when they have exceeded the boundaries of the technical debt that is acceptable and when they need to get help from the external part of the organization, the systems engineering or other parts of the organization, to make sure that the architecture can evolve. So, that is the flip side to this.

Mary Ann: That is the flip side. Sometimes the governance or the ideas of what are the boundaries of when technical debt is too much, that comes from the technical team, not necessarily from the externals, because they really know when things are to the point where, *If I make one more change, something is going to break* because they have postponed things.

Suzanne: Well, and your continuous integration cycle often tells you that, right?

Mary Ann: And, *We will show you that*. So, you need to allow that. You just can't say, *Oh no, we'll do that later*. You have to allow that. At a certain point you have to be prepared to include that in one of your sprints, otherwise you are going to be shooting yourself in the foot.

Suzanne: There are techniques for doing that. There are people that have, that program in, sometimes they're called hardening sprints and other activities within a release, so that you don't let that go too long.

Mary Ann: Or, some people allow a percentage in a later release, not the first one because you are not going to have any the first time. But, in the third or fourth release, you might allow 10 or 15 percent for these technical debt issues. And, if there is none severe enough to fix, then you don't. But, you allow that so that you have already got a plan that, *Yes, we are going to at least consider it*.

Suzanne: OK, I think that about covers it on this particular topic although there's always more to say on these things.

Mary Ann: Oh there is, but I think we did good.

Suzanne: I want to thank you for this conversation today.

Mary Ann: You are very welcome.

Suzanne: In the next episode in the series we're going to explore the final Agile principle. *At regular intervals the team reflects on how to become more effective then tunes and adjusts its behavior accordingly*.

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 sei.cmu.edu/library/reportspapers.cfm.

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.