Toward Efficient and Effective Software Sustainment

featuring Mike Phillips as Interviewed by 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 [DoD]. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

My name is Suzanne Miller. I am a principal researcher here at the SEI. Today Mike Phillips is joining me to talk about efficient and effective software sustainment. First, let me tell you a little bit about Mike, who I have known for many years. He is a principal engineer here who focuses on sustaining legacy weapons systems that are no longer in production but are expected to remain a key component of our defense capability, in many cases for decades to come. Not all of them were planned for that long, but we've got to keep them going.

As an Air Force senior officer, he led Air Force program offices development and acquisition of the software intensive B-2 Spirit stealth bomber using integrated product teams, which I know at the time that you did that, integrated product teams were not as big a thing as they are now. He was very innovative even when he was in the Air Force.

Welcome, Mike, glad to have you here.

Mike Philips: Thank you. It is a pleasure to be here.

Suzanne: Software sustainment, we hear about the long tail. We hear all these things, doom and gloom. Why is it important for DoD systems to be really effective in software sustainment?

Mike: I think that is a great question. It is also a reflection on how things have changed from the earlier days to today. That being, we now put so much of our capability, we stuff capability into ones and zeros, into software that populates, that is, in essence, the essence of the system. Then we put it up into the air or under the water or across the desert in ways that the Department of Defense particularly needs to be effective with that system.

Suzanne: And robust.

Mike: Robust. Not only do they need to be robust, but they need to be able to keep reusing these capabilities in many different environments and over extended periods of time.

Suzanne: And responding to different threats. The threats change. The environments that we are primarily operating in change. In Vietnam, it was jungles. Iraq, it was deserts. Those are two extremes. The physical part of the system has to evolve, but the software part is, as you said, we stuff more and more capability in that because it is the virtual part. It is the part that does not have to actually expand in size physically to be able to get more.

Mike: I think for those that are not particularly familiar with the military side of things, we see the same thing happening with our cars. We now have cars that—not many, but it is growing—cars that will be getting updates…

Suzanne: Oh, I have one of those. I get a USB drive that plugs in, and that is how I do software updates. I pray that it turns on again when I push that button. After the update, just like computers. We are really in the space where we have to worry about all those things.

Mike: That is right. Certainly for many of our systems—we speak of robust— so part of the advantage that we should also recognize is that our ability to build the hardware sturdily, that is rugged in ways that are pretty remarkable in itself. Because of that, we can reuse these older systems.

I find it fascinating that one that was created about the time I was born, the B-52 bomber, is flying now, and is now expected to continue in operations for perhaps another 20 years. There are stories today of grandfathers saying that they flew something that now their grandson or granddaughter is flying that airplane—perhaps not exactly the same one, but the same design—many, many years later.

Suzanne: I know of cases—this is in the commercial airspace—where you change out the cockpit which is where a lot of the software intensity is, but you are using the same airframe. You are using the same physical system...

Mike: Physical system underneath.

Suzanne: That is becoming common. I think the military may have been ahead of the commercial space a little bit in terms of having to deal with that.

Mike: That is right. Because of the other things that had happened around it that created more of that.

Suzanne: One of the things this means is that the DoD in particular has more experience dealing with this problem than some commercial settings. You are one of the people that has studied what works and what does not work.

Mike: That is right.

Suzanne: That is what the blog post series is about that you have just finished publishing. Give us some ideas about what is it that you observed in successful programs that made them successful in sustaining these long-lived systems.

Mike: Well, that might be a point to mention. There are two reasons, two vectors that both led me to this point. One was, as you mentioned, my predecessor time in the Air Force was with the B-2. In that, we had four major teams all working under one two-star general. I had responsibility for the development. However, another colonel, a great peer and friend, was responsible for sustainment. In that sustainment was what became a major site; we now call that particular site the *21st B-2*. There are 20 operational airplanes. Then there is this collection of stuff on the ground that is an integration laboratory. Everything that goes on, any of the 20 airplanes must first traverse through that collection of stuff to make sure it is going to work before it gets into flight.

Suzanne: It has to work on the 21st airplane before it can go on any of the others.

Mike: That is right. That was the one vector. Now, the other vector was at the institute. I have been associated now for a number of years with a thing called CMMI that we have now transitioned out.

Suzanne: The Capability Maturity Model.

Mike: In that work we found that many of the organizations that were actually leading the way in software process improvement were within that domain of sustainment, software sustainment organizations.

They were using that to make sure that they had the competence they needed for all of these things that would be transitioned to them from the prime contractors who were going to make the first thing. But they were then going to get responsibility for care and maintenance of that and often future versions.

Suzanne: Enhancements.

Mike: Enhancements. Those enhancements often were not appreciated by those of us that watched the old system where we thought of enhancements being something [like], *Oh, well that will be a whole new system*. Well, no, it will use, often, the old hardware system but for that

reason, in many cases, needs to be able to be accomplished by, I will call it the *organic side* or *the side that the overall system has been transitioned to,* to take care of.

Suzanne: I have worked with some of those organizations too. They have a huge challenge because they are taking things from multiple contractors. It has been integrated, and it is been tested. It has been certified, but as soon as you make any change anywhere in that system, now you have got to understand everything to be able to understand what the effects of that are.

These kinds of sustainment organizations are really the guts of competence related to what this system is really all about.

Mike: Correct.

Suzanne: Being effective in that role is something that takes a lot of, it takes a lot of time, and it takes a lot of practice and tools, and all the rest of the things.

What are some of the things that distinguish, in your mind, the best sustainment organizations that you have seen? What makes them different from others that just barely hang on?

Mike: Well, the ones that struck me, again, there is this correlation, but when the leads of the organizations were in fact champions of change, effective change, they were looking for ways to make their capabilities better. What that breeds is then people are willing to come work for a place that has that kind of vision.

Suzanne: So it is back to leadership? Vision? And able to sustain a pace of change of the things that we know are the human things that are really difficult to sustain over a long period of time.

Mike: Right. The collection of the blogs to date have covered the Air Force with several of those organizations that have committed to that kind of constantly taking advantage of the opportunity to improve how they do things. That in turn builds a collection of capabilities, and we will be talking about some of those a little bit later as well. That seemed to be one of those key ingredients for it. It is not limited to one service. Certainly I paid attention first to the Air Force side because that is where I came from…

Suzanne: Sure, that was your home.

Mike: But I have seen the same kind of evidence in each of the services, and that is very refreshing to me to see it. It also is significant that a lot of what I am seeing there is not limited to, "*the people that are in the government*," but they have often brought in under contract approaches, teams within, that helped keep that going.

Suzanne: In the settings that I have seen where that worked really well, one of the things I have seen that I would say distinguishes those organizations is that they are kind of badge-less in the sense that you may come from three or four different contractors. You may be civilian-military. You may be military. If I did not know from looking at your badge where you came from I would not know. You are just all part of the team for that program. When I see that, that is one of the things that I think distinguishes a lot of these really high-performing organizations that are really focused on the mission.

Mike: Yes.

One instance that you talked about is the Software Engineering Directorate in Huntsville, Alabama, which I also have had the privilege to do some work with, and they are the home, for some of our listeners may not be aware that that is really the source of the predecessor to the Architecture Analysis Design Language (AADL), which is now a Society for Automotive Engineering [SAE] standard. This came out of a sustainment need, and it came out of an Army directorate.

Suzanne: So why don't you tell us a little bit about that sustainment situation?

Mike: I am going to say two things about that. First of all, we have some excellent blogs already about AADL and the particular program…

Suzanne: And podcasts.

Mike: That is right. The particular program that I will be mentioning, the Apache, was within that domain. The reason I want to bring it up from the standpoint of this kind of conversation about sustainment is that where this was being applied was not in essence a new airplane, or a new helicopter in this particular case, but an evolving one.

In fact, what was occurring was that the version that caused a lot of this interest was the D-version of the Apache, which is now being converted as a released thing from the D-version to an E-version because of the software contained. So the intensity of change and the growth in software capability is remarkable. They are, in essence, retrofitting an existing air frame, the D model, making it then an E model with much expanded capability. Now, how does that get done? Well, that means you stuff the airplane with more new tools, more new ways to work. *Avionics* is a funny term for a little bit of metal surrounding a whole lot of software. In each of these cases, those sorts of things needed to be installed, often coming from very different suppliers.

Where AADL was such a remarkable find was it is a language that sits above the individual pieces and helps you observe what is going on there. I think what is particularly remarkable. Now, one of the findings that we have had, is that AADL always made sense that you need

something, an over-arching approach to bring architectural approaches in to better understand ability.

Suzanne: And alignment.

Mike: And alignment. In addition, we are discovering how well it can assist in things like security and safety concerns. Whereas if you are down at the software level looking at lines of code and trying to discern from that investigation what is likely to happen, it is very, very difficult.

AADL seems to be helping in many ways. A little bit of an advertisement for thinking about for having our audience think about AADL from a sustainment standpoint. It is a recognition that these new ideas, and AADL being one of them, can be employed by groups that we normally think of being at the tail-end of the system.

Oh well, they won't be leading organizations. They will be the trailing ones, the followers. Well, that is not the case when you have a technique like this that very much enriches their ability to do their job.

Suzanne: One of the things that I think AADL brings to the table that is important when you are thinking about evolving a system, is that that language allows you to have multiple levels of detail of knowledge about different pieces. *The piece that is already built, I have a lot of detailed knowledge about. I can put that into the language and show what the effects are there. The new thing that I am trying to add in, at first I am not going to have a lot of knowledge about it, but it still allows me to add that into the model, put what I know in, put in my uncertainties, here are the things I do not know*. *Then it helps me make decisions*. Having all that detailed knowledge from the existing product, which is stable—that is a great gift. If you have not thought about AADL for sustainment, that is, I think, an interesting connection.

You have played with process stuff. You have played with modeling stuff in relationship to sustainment. What other things are you looking at in terms of ways to improve the sustainment posture, both in the military and eventually commercial?

Mike: Well, it is intriguing that we have had a couple of challenges that we have been able to respond to. In one case, it was a particular program, the B-2, that asked us to look around and look at all of the kinds of sustainment that was being done elsewhere beyond the one that they were doing at Tinker Air Force Base, where their site was located.

Suzanne: Where the 21st is, yes.

Mike: It was interesting to see the richness of different approaches that were being taken from base to base in the Air Force. About the time we were doing that, there were a couple of studies.

In the blogs related to this work you will see those particular ones called out. We had membership in two of the significant ones looking at sustainment, both of which said *We need to start thinking more successfully about software sustainment*.

One of the things that often gets missed is that typically when someone says, *Oh, we are in sustainment*, they think about the fact that all of our major systems go through a retrofit kind of process about every four to six years, something like that. There are those who say that that is kind of shining up the metal and repainting and doing things like that.

Suzanne: Which are all necessary things, but it is also looking at metal fatigue. There are a lot of things that on the hardware need to be refreshed and examined. Maybe not refreshed, but at least examined.

Mike: That is right.

Suzanne: If you do not change the software, you do not have that need. But as soon as you make any change to the software, you have a whole bunch of potentially unintended effects that you have to deal with. That needs a retrofit for a cycle as well.

Mike: The other thing that happens that I think does not often get sufficiently appreciated is that software sustainment has a different role. We do not change software just to polish it up, just to fix it. We change software, in fact, to give us new capabilities.

Sometimes those new capabilities are because we have encountered a new way of operating the system that the old software did not adequately do. That would come across as being a problem report. It sounds like the software broke. Well, we asked it to do something…

Suzanne: It is really about the world changing. *We need to do something new, and the software was never intended to do X, but we need X now. Therefore if the software does not give us X, then it is wrong.*

Mike: Exactly. There is a term that will probably be used for many reasons, but we call it software maintenance. But most of the things that get done to the software are not what I think of as maintenance. They are development. Recognizing the richness of development approaches, of ways to integrate new and different approaches in, and even in some cases to rearchitect the system.

In the blog, if someone were to read it, you would see a mention of a particular thing associated with what used to be called the *multiple launch rocket system*. The prime contractor had developed a collection of software modules, and it had grown to the point that there were like 30, roughly, 35 modules that all had to work together. This was getting to be difficult from a

maintenance standpoint to make sure that when you press the button to test it that everything would work out right.

So, in fact, the sustainment unit, it happened to be at Huntsville, got challenged with *We need to reduce the size of this thing because the next box that we put this software in, it needs to be smaller. We need to rearchitect…*

Suzanne: *We need to simplify. We need to look at what we need today as opposed to what we needed 10, 15 years ago.*

Mike: In essence they have refactored all of the software in there down to a size that was about a seventh the size of its predecessor. In doing that, we have also shrunk the time when the operator goes to say *I have got to do a quick check* that is happening now in a couple of minutes as opposed to 10-to-15 minutes.

Suzanne: So they have improved the performance of the system as well as simplifying.

Mike: The performance is going up in that sense, OK?

Suzanne: Some of our listeners may have listened to another blog, that blog post podcast that we did about avoiding complexity, avoiding non-essential complexity. That is a different example of what you are talking about today.

Mike: It is all in that notion that a lot of times when we do these things we say that we are simply trying to get at the heart of…keep the same requirements but we need to restructure it to lead to that more simple approach.

Suzanne: We need to avoid unnecessary complexity. That has a lot of effects if we can do that. I think you are going to be busy. I think there are many systems that still have these sustainment issues that you are addressing.

I want to thank you for joining us today and talking about this, and I look forward to some more work in this area, more blog posts and your insights. I think you bring a very practical side to this that we do not see as often, sometimes, as we would like. I really appreciate that.

The blog post on software sustainment are on insights.sei.cmu.edu. If our viewers would click on the button in the upper left-hand corner and find Mike's name, which is Mike Phillips, then you will be able to look at all these blog posts and other things that he has written.

And wherever possible we will include links to the resources that we mentioned in today's podcast, which is available on the SEI website at sei.cmu.edu/podcasts. And do not forget we are also on Carnegie Mellon University's iTunes U site. The video of this podcast will also be posted

on the SEI's YouTube channel. As always, if you have any questions, please do not hesitate to email us at info@sei.cmu.edu. Thank you for listening.