The Role of the Software Factory in Acquisition and Sustainment

featuring Dr. Paul Nielsen as Interviewed by Suzanne Miller

Suzanne Miller: Welcome to SEI Podcast Series, a production of Carnegie Mellon University's Software Engineering Institute. The SEI is a federally-funded research and development center operated by Carnegie Mellon University and sponsored by the U.S. Department of Defense. A copy of today's podcast will be available on the SEI website at sei.cmu.edu/podcasts.

My name is Suzanne Miller. I am a principal researcher here at the SEI. Today, I am thrilled to introduce our director of the SEI, no other than Dr. Paul Nielsen.

Paul Nielsen: Yes, nice to see you, Suzanne.

Suzanne: He has been our CEO since 2004, so has grown up with a lot of the things that have been happening in the defense acquisition system and the things that the SEI deals with. Today we are here to talk about your involvement in the Defense Science Board that recently published a report—it was a Valentine's Day Report—February 14, 2018, and talk about some particular topics in there. The two that you and I have spoken about are software factories and *software is immortal*, which is quite a quote.

Before we get into that, would you give our audience that don't know you well a little bit of your background and how do you end up with all of this Air-Force-acquisition, Science-Board kinds-of activities?

Paul: Sure, I would be happy to, Suzie. Well, as you know, I started my career in the Air Force. I went to the Air Force Academy and then spent 32 years in the Air Force. What is unusual for an Air Force person is I wasn't a pilot; I was always a research and development person. During the years I was in the Air Force, from 1972 to 2004, systems changed a lot. In 1972, systems did not have a lot of software in them, but by 2004 they had lots of software in them. I watched this whole thing grow during my Air Force career.

At one point in the early ’80s, I was working on a satellite program. We had software on board and software on the ground stations, but the software on board was considered a small component at the time. By the end of my career, software often was the whole system. That is where the functionality was. Big airplanes, tanks, ships—most of the functionality in the systems is really being determined by software.

Suzanne: One quote I saw is that 80 percent of the modern airframe, its functionality, is derived from software.

Paul: That is what they say, yes.

Suzanne: You can’t land it without software. You can’t take off without the software, and you can’t navigate without the software. So, we are software-reliant.

Paul: That’s true. I was trained as a physicist, not as a software engineer, because when I was going through school they didn’t have such a thing as software engineers. I was always entranced by the computer science side of this. When I did my dissertation it was a computational physics kind of project. I was big into modeling and simulation at the time.

Suzanne: Which has come back again. We will have another conversation about that sometime because that is a topic all unto itself.

So, the Defense Science Board looked at defense acquisition of software-intensive, software-reliant systems most recently. You were part of that team. There were several recommendations that are very interesting out of that. The first one is that we need to fundamentally adopt the concept of the software factory.

Now, you and I have both been involved in software long enough to know that the software factory idea has been around for a long time and had some connotations in the ’80s. I am not sure that we are talking about the same thing today.

Paul: No, we are not, thank goodness.

Suzanne: So I want you to talk about what are we talking about today, and how is that different?

Paul: A good point because when people first hear the words *software factory*, they think about that 1980s or maybe early ’90s view. That was more like a modern industrial factory, you know, with people almost in cubicles going *chunk-a-chunk-a-chunk*. That is not what we mean by a software factory now. What we mean is really building the environment around the programmers to help them be more productive, to help them do their job creatively—because they are doing a very creative job when they build software—but do it with support systems that help manage the configuration control, that help test stuff every day, that look for bugs and cybersecurity flaws.

Now we have so much capacity in our computers that we can do this every day and test our software every day as it gets checked in. This allows people to be much more productive, and if there is a problem, you find out right away, so you can learn from your problem rather than the more traditional approach where maybe the major test was not done for six weeks, maybe even six months, at which time the nexus for the creation is lost.

Suzanne: I don't know about you, if you ask me, *What did I write in a report six months ago*? that is a little tough. If you ask me what I wrote yesterday, that is a lot easier. A lot of what we are talking about in terms of the support that the software factory provides is exactly what you are saying, that support so we can do continuous-almost feedback. That feedback is not just from the test system or the configuration but also allows us to actually get users involved, right?

Paul: Exactly. As most people know, this is the [model] Silicon Valley companies, even those companies up in Seattle like Amazon and such, follow. They have done it to great effect. This is one case where the military or the government can learn from industry, sort of a spin-in to the government. The government has traditionally followed other approaches that were very requirements-based. They have perfected requirements engineering. What we have found is that in many cases with software systems, we really don't know the requirements when we start, not completely, and they evolve with time as users start to experience the software.

Another whole point of the software factory is to really enable continuous delivery, continuous integration, which allows users to interact with this much more frequently to figure out *is this going in the right direction? Do I have something that's usable now? Oh, what new feature would I like on top of that*? It is a much more intuitive and aggressive way to develop software.

Suzanne: And collaborative.

Paul: And collaborative. Yes, indeed.

Suzanne: Any of our listeners probably that are in software development in Silicon Valley and other places probably recognize that we are talking about a lot of the principles that come out of the *Agile* development sort of thing. That is another piece of this, that the adoption of continuous iterative (may or may not use the word *Agile*) but continuous iterative collaborative software development is really interleaved with this software factory idea. You are not looking at a command and control software factory, which is kind of what we were talking about in the '80s; this is meant to have that same basis of Agile principles that drive why we are doing the feedback.

Paul: It is one of the unfortunate implications of the word *factory* sometimes. It sounds like we are trying to drive out the initiative and creativity, when what we are really trying to do is enable the initiative and creativity of the programmers.

Suzanne: One of the things I used to tell people when I worked in process improvement consulting is, *We are not trying to make things that should be creative routine, we are trying to make the things that should be routine, actually be routine. We do not want configuration management to have to be something you figure out every time you check in your code. We want that to be routine, we want you to be able to figure out the clever algorithms that are going to get the result that your customer wants. But we want these other routine things to actually be easy.*

Paul: That is exactly right.

Suzanne: Thirty years after the initial concept of the software factory, the tool chains that are in place now are really one of the enablers of this. The tool chains are enabled by the processing power. The processing power is enabled by all of the innovation, a lot of which comes out of the defense industry. But all this goes back, but it has taken a long time to get to a point where we can actually do this.

Paul: Well, sure, and the other thing that has really happened is the general availability of cloud services allows us to build these tools in the cloud so that we can test them easily at night time with lots of computational power, lots of memory, and such.

Suzanne: One of the things I noticed in the report, speaking of virtualization and cloud, is you do address some of the unique issues of cybersecurity because there are cloud environments that are a little too open for the Department of Defense. But we also have some concepts that have been very useful that we can take into the cloud. The software factory idea is not negated just because we have to be in a secure environment.

Paul: That's right. The other thing for our listeners to remember is that in most cases, the Department of Defense does not develop its own software. We do have some places where real government people are writing software, and they do it very well. But most of this is being done by the defense industrial base, by the major corporations that you usually associate with the Defense Department: the Lockheed Martins and Boeings and Raytheons and companies like that. So this is not something that the Department of Defense can do by itself. It is something that has to involve the whole defense industrial base as well.

Suzanne: That is something that we are also then following to a certain extent—the workforce-development kinds of things that are going on in the defense industrial base because commercial is educating software engineers. Our education system now is educating software engineers in these techniques. So, the DIB, defense industrial base, is getting those engineers in. So, exposing them to the waterfall processes after they have been taught…

Paul: It's not a good fit.

Suzanne: Not so good.

Paul: Not a good fit, it doesn't excite them; it makes it hard to recruit them, frankly. Because they know there is a better way to develop software, a more exciting way. I first learned to program 52 years ago. I know I look too young for that.

Suzanne: You do, absolutely.

Paul: But 52 years ago, it was FORTRAN at the time. In a sense we kind of intuitively did an iterative design in those days because we would build a little and made sure it worked, and then we would build a little more and…

Suzanne: Because it took four hours to get it turned around.

Paul: That's right. That's right.

Suzanne: Yes, I mean, that's—yes.

Paul: This is a return to our roots in some ways albeit with lots of great tools and technology around that to make you much more efficient.

Suzanne: And faster. I mean, the iterative process in the '70s was based on actually processing power restrictions. *It takes four hours to get your program back and your results back, right? So I can't afford for it to be wrong.* Now we actually have this ability of almost continuous feedback from compile all the way to regression test. So, that is a big difference.

So let's talk about the flipside of that. We can be very very fast with our feedback for software that—and I love this phrase—*software is immortal*. That is a recognition that I am not sure even the commercial industry thinks about that much. We tend not to say, *OK, we are done with this piece of software, let's build a new one*. We just continually evolve the software that we have, unless it becomes so brittle that we have to replace it. If we think about it differently—that this software is going to be around forever— how does that change the way we think about sustainment, contracting, engineering, architecture? I think architecture is the big one in that sort of thing.

Paul: Well, to go back to my Air Force and DoD roots, there was a time when we formally would develop programs, and then we would have something called program management responsibility transfer. We would hand it off to logistics and sustainment organizations. That kind of stopped, even for hardware-intensive programs, in the 1990s, mid-1990s, because we started to recognize that there was a more continuous life cycle development, and so we went to a total life cycle development kind of process.

Suzanne: And integrated logistics.

Paul: That's right, integrated logistics.

Suzanne: Meant to be integrated into the development process.

Paul: Even then, in a hardware-type system, there is a design phase, a build phase, and then maintenance. What happens in software is that software never stops evolving. Even on your cell phone, you get those little notices that you have an update for your app, right?

Suzanne: Yes. All the time. I pushed *later* just this morning.

Paul: Or that an operating system update comes, which takes a little longer to do. What we are seeing is that there is no good reason to consider the software to be in sustainment. Because, not only are you going to try to do those fixes, but you also want to take advantage of the flexibility of software to be updated and improved throughout its life. We don't want the software in 2030 to be the software of 2020, right? We want the software in 2030 to be even better and to do more functionality. One of the great enablers of software-intensive systems is that these systems can evolve at software speed, not at hardware speed.

Suzanne: Oh, I like that.

Paul: We can really keep these systems up to date and give users new capabilities on weekly, monthly—or whatever kind of term we think is right—basis as opposed to maybe a 10-year block change on a big airplane or a 20-year block change.

Suzanne: We are looking at these methods and looking at this perspective for both, sort of what I call traditional IT systems—the payroll system and some of our climate control—but also we are looking at it from the viewpoint of complex airframes and satellites and trying to look at how do we bring the lean engineering ideas into the hardware side also. Because if we think about this as an immortal software, B-52's are 70 I think.

Paul: Yes, they're getting pretty old, yes.

Suzanne: We have some very old hardware.

Paul: Yes, and they are going to be fine for a few more years, too.

Suzanne: We have done this. We have proven that it is more or less immortal in hardware too. So, we have got to deal with those things.

Now, those are things that are kind of related to our current systems in a lot of ways. There is a new kind of system that we are starting to see deployed more and more frequently. These are the systems that really take advantage of what we used to call AI [artificial intelligence] again, in the '80s, but we call machine learning now. The autonomous kinds of vehicles and systems that are operated where their learning actually evolves that.

The basis for those—you and I learned FORTRAN—so very procedural, but these are much more mathematical algorithms, more than, *How did you code this procedurally*?, or even from an object-oriented viewpoint. *How are we going to verify those systems?* That is an old-but-new-again question.

Paul: This is a very tough question, and it applies to learning systems that maybe stop learning after development, as well as learning systems that continue to learn after development. Because one of the problems is that using these neural nets or deep learning kinds of systems we don't really know how a system made its decisions ultimately. We can test it and see what it does, but there's always that one odd test that it makes a different decision on.

Suzanne: An edge case.

Paul: An edge case, yes. So, we are going to have to come up with different ways to test and ways to continue to collect data even as systems are in the field. This is one reason why a lot of the self-driving car manufacturers are out there not just designing inside their factory but also operating stuff on the economy to collect data to see what the real world does. It also involves sometimes a lot of modeling and simulation to build even more data in.

Sometimes in defense systems it is a little harder to collect data from the field. We don't have the millions of drivers that the automobile world has. We don't have the millions of users that an app would have. When we do use systems, sometimes they are in very serious and potentially lethal consequences. So, we have to find some ways to have assurance that these systems are going to work the way they say they are, the way we plan them to be, when we use them for the first time and when we use them afterwards.

Now, in the case of a system that continues to learn, it is not the same system a week later than it was the week before. This is not the way the Defense Department has traditionally done verification and validation, where they like to test a system of record and then release it to the field. These systems are going to continue to evolve as they go out in time. So I think what this portends is that for the IV&V [independent verification and validation] community, the independent test community, as well as the development test community, they're going to have to work much closer together, throughout the whole life cycle of the system. So, the independent testers sometimes haven't been involved in the very front, and sometimes the development testers haven't been involved in the very back. But really, they are going to have to meld their interest and their procedures to collect the best data for everybody.

Suzanne: In some ways I could argue that what you are saying is the idea of extending the software factory into more of test and operations, which gets us to another concept that the DSB report talked about, which is DevOps, this idea that we need to have a confluence of development and operational concerns all throughout the entire development and sustainment and operations lifecycles. So all these things come together.

Paul: Of course, one of the big concepts that the DevOps movement brought in was the whole idea of a minimal viable product. That has worked really well for the Silicon Valley companies, for the companies that are building on the Internet infrastructure. That can work really great in the IT systems. We have to find ways to take that into the embedded system world, because in embedded systems we don't always have the platform right away, but we can at least simulate the platform and develop capabilities in a similar environment even as the hardware is coming along as well.

Suzanne: We are seeing much more interest in that. One of the pieces of work that we have been doing at the SEI for several years is model-based engineering and the languages that support that. That is another area that we need to support better infrastructure in. I am seeing huge changes in the last couple of years in the interest in that technology. So there are things that the SEI, in some ways, has been the previewer of some of the things that are coming on now. We are also very involved in the DevOps world, in the Agile world. We are actually in the space that this is talking about; sustainment, you know, *how do we acquire systems differently?*

What do you see as sort of the key elements of the future of defense acquisition that we, the SEI, and the community as a whole need to be watching for so that we can be ready?

Paul: I am really heartened by the response to this study and by some of the things I have seen since the study concluded. because when you go across DoD to the defense contractors as well as to the organic assets in DoD, you see people moving toward this approach. There are islands that are doing this quite well inside the government already. I visited something at Schriever Air Force Base a few weeks ago showing a great DevOps approach to developing software for the National Space Defense Center.

Up at Hanscom [Air Force Base] I think there's a whole team that's working with Pivotal to train people in these kinds of techniques to develop software for the air operation centers and others like that. I know there are similar things in the Army and Navy; I am just not as aware of them, but I can see this happening. The thing that is great about it is the people that are going through it are excited because they can see the results of their work and see it working. When you are on a program that takes 20 years to deliver, you don't see that. People like to see the fruits of their labor and see that they work. This is actually keeping people more excited about being in the defense industrial base, if the base can change a little bit like this.

Suzanne: I am seeing that in our work as well. It's very promising. People like to be able to be connected to their purpose. People don't join the government workforce to become rich.

Paul: No, that is for sure.

Suzanne: We can attest to that. But they join because of the purpose, they join because of the mission.

Paul: They join for the mission.

Suzanne: And being able to see connection to that mission is one of the things that drives people to stay, and we need that talent to stay.

Paul: Oh, yes. It is a mix of workforce too because there is the organic side of the services as well as the defense industrial base, and they both have to be in harmony. One of the things that is a little hard right now is even though everybody wants to move toward this more Agile approach, we still have processes in the government for oversight that are built for the previous way of doing business, and we have to learn how to adapt those best for this.

When you do this, you give a lot of responsibility to the individual work team that is actually building the software. It is not so much top-down as bottoms-up, and so we have to adjust our systems to trust those work teams of the experts who know what they are doing to listen to them when they say what their schedule is going to be, what their problems are, and to give them to creativity space to do what they do best.

Suzanne: There are economic implications to all of this as well as security implications. There is a lot that goes around this. If you were to say what is the one thing that if you are a program manager out in the DoD world right now you should be really paying attention to that reflects what the DSB came out with, what is the one thing that you would say, *pay attention to this*?

Paul: Yes, I think first off you would want to make sure that your contractor—whether you are just getting proposals in or actually on contract already—has an effective software factory environment. That they know what it is.

Suzanne: If they say, *What is a software factory, that may be a problem*?

Paul: Right. That they know what it is and that they are taking advantage of all the kind of tools that are out there. These tools are evolving fast too. So the tools that are the best tools of the day today won't be the best ones a year from now. That's how dynamic this whole situation is.

But it is a system where you want to stay in touch with all the commercial side of this software business, too, because they're the ones pushing much of that. I think also I would really want to put more emphasis on the workforce and make sure that in my organic workforce, the government side, I have people who are at least savvy about this. They might not have to be the experts because they may not be writing the code, but they better understand the principles and the way to oversee and manage projects that are being done like this.

Suzanne: One of the examples you gave in the DSB report talked about one of the agencies that has adopted this very fully, that they have had to really bring up their game in terms of [having] expertise locally on the government team. We are also seeing that. Where we see success and Agile development is when the government actually knows how to interpret what they are seeing.

Paul: I think it does put a little bit bigger burden on the government to make sure it has technically competent people and in the right numbers too. The government has some great people, but sometimes it hasn't had them in the numbers that they really need to cover all projects.

Suzanne: I want to thank you so much for having this conversation. I think it is important for our viewers and listeners to understand. Go read this report. It's a good report.

Paul: Yes, it's a good report. It's available.

Suzanne: It really is. It's publically available, and the reference to it will be in the podcast transcript, so you can go get it. There are great quotes in there, little stories. I am really pleased that you were able to be a part of this work.

Paul: It was an honor to be a part of that.

Suzanne: Somebody said we have been doing these kinds of reports for 50 years. This is the first report where the examples I saw showed change. In a very significant way. And so, I am very heartened…

Paul: That is great.

Suzanne: That we are kind of on a cusp of a new era perhaps.

Paul: Well, when you work on one of these reports, you hope that people are heartened.

Suzanne: Yes, not depressed, right.

Paul: That they really adopt some of these principles and that it makes an impact. I am really glad to hear that, and I am really happy to have spent some time with you.

Suzanne: Thank you so much. So, for our viewers, this podcast is, once again, available on the SEI website at sei.cmu.edu/podcasts. It is also available on iTunes. It is also available on SEI's very own YouTube channel. And so I encourage you to go and get the transcript, get the resources, and take full advantage of this report. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you for viewing.