Why Software Architects Must Be Involved in the Earliest Systems Engineering Activities

Featuring Sarah Sheard as Interviewed by 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. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Eileen Wrubel: Hello, my name is Eileen Wrubel, and I am the technical director of Transforming Software Acquisition Policy and Practices here at the Software Engineering Institute. Today, I am pleased to sit down with Dr. Sarah Sheard. Dr. Sheard is a principal systems engineer here at the SEI.

Today we are here to talk about why software architects must be involved in the earliest systems engineering activities. Welcome, Dr. Sheard.

Sarah Sheard: Thank you.

Eileen: You recently wrote a blog post on this topic, and it went viral on Hacker News, which was really exciting for all of us. That is what we are really here to really talk about today. Before we get into that, I would like to do what I call stepping back into my way-back machine a little bit. I would like to get to know a little bit about your background. You are a systems engineer. Can you tell me how that field has changed since you first entered it?

Sarah: Thanks, Eileen. I started systems engineering back around 1980, and I was working on satellites. I also have a blog post on how the satellites have changed since then. In general, I think systems engineering kind of jelled as a field in, I'd say, about the 90s. A systems engineer from this company or that company would be able to talk to another systems engineer, and they would be speaking the same language. In that time, of course, software was making this incredibly huge change from something you might write a couple of lines of code for a program to figure out something to the way everything is run.

In that time, the people who were young back when I was young did not work with the software that we have today. These are the people who are running programs now. When they were back being systems engineers or in some other engineering field or even managing at that time, the ground floor of what was being done is different from today. Now that they are managers and at the 50,000-feet level, they are seeing the big picture, but what they have in their head as to what, how things work is really not the same as how it's working today. The people who are doing software, in particular, have developed an enormous number of specialties and approaches that are appropriate for today's technical systems, including that idea of a software architect who defines the basic principles behind how software is going to work.

Eileen: OK. Let's talk about going back into defining those rules for how the software is going to work in the system. When the government, when the DoD needs a capability, how do systems engineers and acquirers typically determine how that problem should be solved and how that capability really is provided across the system?

Sarah: Well, that is a really excellent question, Eileen, because somebody has to make the first cut. We can't take a problem that the government is having, like, *We don't have enough resolution on this particular kind of thing, and we would like to have twice as good a resolution,* and just throw that at the industry. Instead the government itself has to decide, *What is the problem? What kinds of technologies that we have available or that might be available in the near future might help to solve that problem? How are we going to acquire the pieces of technology, so that we can put it together to have the functionality that we need?*

Somebody up there in government land decides how this system is going to be cut up into pieces that can be then allocated out to this contractor, that contractor, and the other contractor. If you are going to build a house, you would probably have a plumber, and you would have an electrician, and you would have somebody working the internet capabilities, the windows people. We all understand with a house, the kinds of pieces you have to cut it into for it to work. But these super new, super high-capability programs that have a lot of software content are making new steps. They are really learning a new way to do things.

Now, the obvious way—and I am going to go back to the example of the global positioning system or GPS. You have got a satellite up there that is taking pictures and understands where it is with respect to the earth. You have got ground stations that run the satellite and also provide a lot of data back and forth. You have got handheld user units which, in this case, is my smartphone. The easiest and most intelligent way to split that up is to have somebody build the satellite, somebody build the ground station, and somebody build the handheld systems. It seems to be utterly clear and obvious that you should do that. But there is a problem. It worked when the primary difficulty of engineering was in building the thing.

Is the satellite going to be stable at this orbit? Is it going to get enough power? On the ground station, do we have enough input, output? Do we have computational capabilities? Is it going to survive through weather, et cetera*? The handheld units, are they going to break?* Whatever the hard things about the units used to be the problem. It is kind of not the problem anymore because we have a lot of that down. The bigger problem is the software that provides the continuing capability from yesterday continuing into tomorrow, but also anything new that is useful to people is generally based on a new kind of software. Now, the software exists because things need to be communicating with each other. It is inherently an inter-piece of function that the software has to provide. If you are going to do that, you have to watch out that the software isn't split into pieces.

Eileen: That seems to set the stage for what kinds of problems we can really have if a systems engineer doesn't have experience, expertise, breadth in software.

Sarah: That is I think exactly where I was hoping we would go in this. The folks who are in the early system design phase of programs—in other words, before contracts are even let out—tend to be pretty senior people like myself. They have not been on the ground floor of building software today. Because if they were, they would be more junior folks. They don't necessarily know how the software is working and how a software architecture works and what the difficulties are, particularly with newer and newer systems. This goes for not only the people who are doing, I would call their role is more of acquirers, but also for the systems engineers that then come and help them, who also tend to be senior.

They don't necessarily understand, *If we break the system into these pieces versus those pieces, what are the benefits, pros and cons, with respect to software?* They would understand that, *This particular thing would be better in wet weather for the hardware. This one would be longer lifetime, and this one would be smaller, and this one would be cheaper,* but with respect to the software, it's kind of throwing things up in the air.

What we tend to do today is we say, *You are going to build the satellite and all of the software that has to do with the satellite. You are going to build the ground station and all the software that has to do with the ground station. You are going to build the handheld unit and all the software*. What happens? Well, what happens is the software built for the handheld doesn't communicate well, rapidly, effectively, efficiently, ingeniously with the other software. Then, the software is broken. Remember, this software is creating…. Most of the new functionality and most of the value of a new system is the software. You have to allow for that software to be architected efficiently and well the first time if you want it to be able to do that effectively. Does that make sense?

Eileen: It does. You want to make sure that you are allocating functionality to the hardware and the software in a way that is optimal for the platform and not just one of those concerns independently.

Sarah: That sounds like a problem because we do need the satellite to have its software. In particular, anything it needs to allocate power from the solar cells to the different units, the different parts of the year, like eclipse or where it is colder or warmer. We need the ground station to have its software work, and the handheld to have whatever it needs to do work. Most importantly, we are forgetting that some of the software has to communicate all three together, and it has to make sure that each of the functions of the phone works with all the other functions.

To use an example that was given to me by our former CTO, they were looking at why it took so long to make certain computations on a battlefield system. It turned out that two or three different contractors were involved in the process of this computation. He said that when they looked into it, they found out that there was a Fourier transform back and forth between the frequency and time domains between this point at the beginning of the computation and the end of it, five times. Nobody had decided what was going to be done in the frequency domain versus time domain. Everybody got the data in from the last phase and said, *Oh, I have to switch it around and do this computation*, which not only slows it down but, as you know, it adds noise, and it adds a lot of uncertainty to it.

Nobody had taken the software point of view up front saying, *What are the parameters that we have to settle on? What are the most important computations that are going to occur on each of these links? How are we going to architect it so that we could get this parameter that we care very much about in terms of a quality attribute of the system? We sacrificed this one, which we don't care so much about*. Because, as you know from software architecture, it is all about setting these quality attributes. *If you set it one way, you will get these quality attributes over this one. If you set it another way, you will get these*. So, the systems engineer has to know what they need out of the system in terms of what will make it function properly. But the software architect is the one who is up to speed on how software works, understands the changes that have occurred in cybersecurity, in software technology, in databases, encryption, whatever, and understands what kinds of software architectures will provide those most needed and most important system attributes. So that they can develop the software to prevent having computations take five different long steps when they could just be a passthrough or whatever it was that they needed for that computation.

Eileen: This early engagement also helps us into that trade space where we are allocating functionality between software and hardware when we have a choice.

Sarah: Right.

Eileen: There are some things we can choose to do in the physical realm or in the software realm, and we have to make choices based on those quality attributes in system performance objectives about which implementation we are going to pursue.

Sarah: An example of it that came out in my work was we were looking at, say, a battlefield drone. Let's suppose this one just takes a picture of the ground. It is simple. We couldn't do this 20 years ago, but today, it is a simple drone. It just takes pictures, and it has to bring that data back to the ground station. Now, does that drone include the functionality to encrypt that data or not? Because if it does, you have got a heavier drone, but you have got a smaller bandwidth because the data that comes down is a smaller amount.

If the drone does not encrypt, then you have got raw data coming down, so it is a bigger bandwidth but a lighter drone. All these have to be traded off, including the fact that this is a battlefield situation. If you have got raw data coming down unencrypted, the enemy is going to have as much access to the data as you are. Do you really want that? Including the fact that if it is encrypted, the people on the ground who are the good guys, us people, they have to have the proper security clearances to use, to be in possession of and use the unencryption material, and it is heavy. They have to get it there as well. So there are a lot of considerations that go into it. It could be that both options have to be carried up to a level at which point you can decide, *In this kind of situation you use this one, and [in] that kind of situation you use the other one*. Sometimes you don't know in advance how a war is going to proceed.

Eileen: We talked about having these conversations based on functional capabilities we want from the system and certain performance characteristics in the real-world conditions in which the system is going to operate. Now, let's talk about sustainability, keeping the platform if you will, keeping the system alive as it is in operations and until we end-of-life it. You wrote in your blog post, *If the software architecture is not modifiable, then the system as a whole may not be sustainable*. Can you talk about the connections between architecture and sustainability?

Sarah: Well, when I was a kid, a phone was this heavy thing made by AT&T and actually leased to the households. After a while, you could buy your own phone. It was unbreakable. You could have five kids in your family. We could be hitting each other with the phone. Nothing was going to happen to that phone. It didn't need to be sustained. AT&T leased them out. You borrowed back the time and money to make phone calls. Nowadays, everybody buys their phone, but you know that things are going to change every night when you go to sleep. Facebook is going to have an update. Every time I go look at my app store, I have got 13 things that need to be updated. They are updated without my permission, but if I don't do it, they are not going to be cyber-secure, so I have to get them updated. So, the sustainability of the system, including that phone, has a lot to do with whether that software can be updated. It will have to be updated. It is very discouraging for people my age who are in the senior levels, for example, of the government to hear, *Software is never finished*, because they think of things as being like that AT&T phone. I can buy a phone. It will work for now until the day I am done with it. Whereas, today with these extra-high capabilities, things are changing. Our adversaries in warfare are coming up with capabilities that we have to then leapfrog over. In order to do that, we have to be able to update the software. In other words, in order to sustain the applicability of our new systems, we have to be able to update the software, so the software has to be, what was the word, *adaptable?*

Eileen: Modifiable.

Sarah: Modifiable. Obviously.

Eileen: Given all of the issues that we have talked about, when should software people first be involved? And we are talking about a contract for a major systems acquisition? That is really up at the front, the very beginning that you are talking about, right?

Sarah: In a sense, it is. If I was a contractor like Boeing or Lockheed Martin or Raytheon or someone, I would think that that is the beginning of the program, but it is not. The beginning of the program is before when we decide what we are going to put out that maybe Boeing or Lockheed Martin or Raytheon or Northrup [Grumman Corporation] or somebody is going to buy. Before the point where the contract is even awarded, or let out in the beginning, is the point where we just split it up. *Are we going to have somebody do all the software on—remember, we were talking that satellite—or, are we going to have some software allocated to a fourth contractor that has to do with linking all the other software together? How are you going to make this work?*

Of course, that is not easy either, because if you do have a software-only contractor and three hardware systems contractors, you know that that software in the software-only contract part is going to have to interface with the handheld. It is going to have to interface with the software on the satellite. It is going to have to interface with the software on the ground station. All those interfaces are going to be extremely difficult in the sense of having immense numbers of details and tradeoffs. Somebody has got to do that work. If you don't do it early, you have got the unsustainable, unmaintainable, maybe-you-can-patch-it-together software that we see frequently today.

We see programs that are doing fine up until the point where they realize that their software is late. It is late because you really can't write software if you don't know what you want it to do. Frequently, you find out what you want it to do later, after the hardware is at least in some degree of preparation. It is not anybody's fault, it is just the way it works. If you don't have software that's modifiable and the architecture is set up for this situation that you are going to find yourself in, you are going to be having software problems. We have them today. We are going to keep having them. The best way to ensure that they will be minimized is to have that software architect be a required part of this early systems engineering team.

There is another reason why you can’t just make it the same person. Systems engineers have to be broad. They have to know about the Navy, and they have to know about technology of lasers. They have to know about the politics going on around this program in Washington. There are so many things that a systems engineer has to know about. They excel, systems engineers, at the 50,000-foot level and the interfaces, how do they all work together. The software people have to take a different tack. If anybody does, they have to do it, which is they have to be looking at the details. If you don’t get your details right, you don’t get working software.

Furthermore, software is changing unreasonably quickly. It will be even faster next year, not only in terms of the technologies that come up but in terms of the things that we have to do to avoid cybersecurity problems. It used to be a yearly, weekly, daily basis. Nowadays, things could change in an hourly fashion. We have to make sure that the software architect is up to speed on that and continues to be up to speed while the systems engineer is looking broadly. I don’t ever think that you are going to get somebody who has time to do both, much less be good at both.

Eileen: We are talking about systems now with expected lifetimes of 50, 60 years for the capability. So we should not expect that the software is going to be the same actual lines of code 50 years from today. We need to be able to architect platforms to be able to evolve the software as we continue to deliver the capability and sustain and upgrade the physical components over time.

Sarah: Exactly. I think every time you make a change to the hardware, you run the risk of having to change the software to compensate to make it work better. Every time you make a change to the software, you make the risk of having to change the hardware. Maybe it doesn’t work with this new software. Maybe the hardware is old, anyway, and now is the time to throw it away. The two are very tightly enmeshed in most programs and will continue to be. You have to have both of those aspects well supported by people who know what they are doing. The systems engineers have their skills. The software architects have one that I believe needs to be brought in much earlier.

Eileen: Can you tell me what is next for you?

Sarah: Well, I am retiring. Within about a month, I will be sitting on a beach in Florida. Well, that will be two months. I think that what I am planning to do is to continue looking into systems engineering because that is my initial love. All this valuable experience I have gained by working at the SEI on the software point of view and, and technologies and, and things. I hope to push it from the systems engineering point of view and hope to be able to involve you and the rest of the SEI as well as we come up to places where we need both sets of skills.

Eileen: I would very much like that. I have enjoyed the opportunity to work with you over the last number of years.

Sarah: Back when you were a small girl, yes.

Eileen: Yes. Well, thanks so much for joining me today. We will include links to all of the resources we referenced in today's podcast in the transcript. Dr. Sheard's blog post is available on the SEI website at insights.sei.cmu.edu. You can click on the Browse by Author at the bottom of the page and search on her last name, which is spelled S-H-E-A-R-D. Thank you, so much.

Sarah: Thank you.

Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn Radio, Google Podcasts, and Apple Podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please, don't hesitate to email us at info@sei.cmu.edu. Thank you.