Enabling Transition from Sustainment to Engineering within the DoD

featuring Thomas Evans and Douglas Schmidt as Interviewed by Eileen Wrubel

---

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 and welcome to the SEI Podcast Series. My name is Eileen Wrubel, and I am a technical director here at the Software Solutions Division of the Software Engineering Institute. Joining me today is Tom Evans and Dr. Doug Schmidt. Tom is a senior software architect in our Software Solutions Division working out of our Boston office. Doug is the associate provost for research development and technologies at Vanderbilt University. He is the co-director of Vanderbilt's Data Science Institute, as well as the Cornelius Vanderbilt professor of engineering, professor of computer science. Previously Doug served as the chief technical officer of the SEI, and he continues to collaborate with us on various projects, such as the ones we are about to discuss today.

Welcome to both of you. So, before we get started, I would like for both of you to tell our audience a little bit about yourself and what brought you to the SEI and the kind of work that you do here, especially with respect to DoD programs. I will start with Doug.

Douglas Schmidt: I have been working with the SEI since roughly 2005. I worked initially on a project related to ultra-large-scale systems (ULSS), was quite fascinating, had a big influence, an impact on the research community, 15 years or so ago. From 2010 to 2012, I was the chief technology officer at the SEI, and since that point I have been working on many other projects together with Tom and other colleagues. What I find most fascinating about the SEI is the chance to work on cutting edge projects that really make the difference in terms of the nation's defense, in terms of being able to make our defense industry based, and commercial companies and government agencies more effective at applying software to meet their mission needs. We are really lucky to have an organization like the SEI to play this role, because software is more and more important, and many other organizations are not able to bridge the gap that the SEI can from advanced research all the way into cutting-edge practice and making these projects work and the technologies work for real systems and real environments.

Ellen: Thanks Doug and Tom can you tell us a little about yourself?

Thomas Evans: Sure, thank you Eileen and thanks to Doug and all the collaborators on this series. I actually grew up in the Pittsburgh area and received an undergraduate degree in computer engineering from Carnegie Mellon [University]. After which I moved to Boston, and I have spent about 25 years in the Boston tech scene. While being well aware of SEI, I never really thought this is where I would be. A few years back I had a particularly frustrating day at a job at a small company that I was hating, and the software there was and still likely is an ancient, tragic mess. My efforts to modernize it were met with a lot of resistance, and I recall thinking to myself, *These folks don't care about quality of this product or how difficult it is to maintain and the fact that it is deployed into some very important public infrastructure situations*. That day I came home from work and decided it is time to go. I logged into LinkedIn and upgraded my membership and the first job recommendation I received was for the SEI. I remember thinking to myself, *Why am I seeing this?* As I read the description, my thoughts changed to, *I have done all of this, and this kind of work actually matters*. So, in the end, I was hired to support a Boston location for SEI and to work efforts in the Boston area. The work I have done at SEI today has involved software architecture and design analysis of Air Force and Navy systems, specifically focusing on modernization of those systems with emphasis on open systems and architectures.

Eileen: Thanks Tom. And so, I am going to build on that. In the last few months, you two along with Mike Gagliardi, Nicholas Reimer, and Joe Kostial, you published a series of SEI blog posts outlining issues that the DoD needs to consider as software sustainment teams transition more into software engineering roles. Before we talk about those issues, I would like to take a step back and talk about how software sustainment in the DoD sense is different form software engineering and why DoD teams are making this transition? Tom I would like to start with you.

Tom: Thank you. So, sustainment focuses on addressing the needs of already-deployed systems. Sustainment usually focuses on the rapid repair of problems that are found in deployment and in operational use. I tend to think of sustainment as brownfield work or dot release work. Whereas engineering is usually focusing on new features and the next big major release. In engineering, the work is much more greenfield, meaning that the field is green and there are no existing structures on it, and teams are making these transitions for a few reasons. First, there is a general shortage of software expertise out there. That fact is only expected to worsen over time. In many ways, it is all hands-on deck, and the belief is there is a short bridge from sustainment to engineering.

Second, especially in DoD work, there is a lot of proprietary software, and that has many non-functional drawbacks, specifically in terms of the proprietary nature of that software. In those cases, it is often the case that only that original contractor is able to modify or fix or enhance those systems. That limits competition and usually ends up costing taxpayers more in the long

run. With these factors, a possible answer is, *Maybe we can do it ourselves within the DoD or at least some major portions of the software can be developed internally and organically*. So, what does the DoD have already is a lot of sustainment organizations. So, the belief is that, *Hey, we can kick start this engineering effort by using these sustainment organizations*. I mean development and sustainment are just the same right?

Eileen: Thanks Tom. Doug, do you want to build on that?

Eileen: Yes. So, about a decade ago when I was the chief technology officer at the SEI, I served on a study at the Air Force Scientific Advisory Board looking at sustaining aging aircraft. One of the things we discovered during our study was that the Air Force was spending a lot of money on sustaining the software portions of these aircraft. There was some concern that at the time a lot of the aircraft that were getting into the sustainment phase were the very complicated ones, like F22 and the joint strike fighter. That was going to cost quite a bit of money using the contractor-based sustainment. So there was a real push on the government to try to move towards so-called organic sustainment where it's done by the government. I think to build off of what Tom said, people are realizing that if the government can do sustainment of complex systems, such as Joint Strike Fighter or F-22, they also perhaps would have the skills and capabilities to do development of these complex systems as well. Part of what we are describing in our studies that we have done and the various blogs that we have written, is the fact that developing software isn't necessarily the same thing as sustaining or maintaining it. You have to take into account different skill sets and different factors and forces that may or may not make it easy to take the same set of people and apply them in one context as successively as you have been able to apply them in another.

Eileen: So, you two have recently written this blog post series, and you talk about the differences in technical perspective and technical focus of developers working in greenfield versus brownfield situations. I am wondering if you can expand on that a little bit for our listeners. I will start with Doug.

Doug: Sure, you bet. So, typical people who develop greenfield systems, the ones that are not yet built, have to have a tremendous amount of knowledge of software engineering concepts, such as software architecture, and software design, and optimization techniques and so on because that they're really trying to create something that may not exist at that point. So there is a certain level of expertise, a certain amount of creativity, a certain ability to work with things where the answers aren't necessarily known at the beginning. Historically, people in contrast to do the sustainment part are given many of those artifacts. They are given tests systems. They are given architectures. They are given detailed designs. They are given everything laid down ahead of time. So their job is just to make sure that those systems continue to work properly as they are being evolved and improved, and that is often a very different skill set. Just in the same way that

people who are great at starting companies, aren't necessarily the same people who are good at running them down the road. People who are great at building systems aren't necessarily the ones who were best served to sustain and maintain them and vice versa. So, I think part of what we are doing in this blog series is trying to help people recognize that there are different skills, and talents, and core competencies between different types of developers working in different contexts.

Eileen: Tom is there anything you would like to add?

Tom: Sure. So, you may not have picked up on my little joke there at the end, that sustainment and development are the same right? I think Doug is pulling that further little bit more for sure. People who do sustainment are often thrown into situations where they don't know the overall picture, and they are able to narrow it down and fix a particular issue without understanding the entirety of the system. Engineering often requires the breadth of knowledge for that entire system. While the skills may be transferable, or transmittable between the two, it is a different mindset and a different approach to look at those two different forms of engineering.

Eileen: What I would like to talk about now is the specific challenges that teams are encountering as they have to make this transition from focusing more on software sustainment challenges and moving more into software and system development roles. In the blog posts, you have broken these into three main areas: people, process, and technology. I am wondering if the two of you can talk to me a little bit more about some of the more pressing issues that you have observed in each of these areas and some strategies for how to address them. I will start with Tom and ask you to maybe to talk about, let's start with process.

Tom: In my observation with process, the biggest hurdles to overcome are what I think of as the rigid interpretations of what is the trendy software process of the day. I often think that process itself is software and should be modifiable by the people involved in that process. I mean I think even in the Agile Manifesto, there was this idea that, process and teams grow organically from the team. What I have often seen is people, especially leadership, will walk into an effort and just throw down, *This is what the process is going to be*. And that may not be what other people have experienced in process. So there needs to be a certain amount of level setting there and determining what team members agree as to the definition of certain concepts like architecture, and MVP (minimum viable product), and what is simple. Because, you know Agile, for instance talks a lot about simple solutions and there's surprisingly a lot of disagreement about what is simple. And so, getting that all setup before you plunge into writing code, is super important. But, more importantly to me and to things I've seen is, remove that rigidity, remove that resistance to hearing what other people think the process can be, or can incorporate.

Eileen: Thanks Tom. Doug, do you want to expand on the process challenges a bit and then maybe talk to us about differences or challenges in adapting to technology aspects?

Doug: One of the challenges that are faced by people doing sustainment is the fact that a lot of the key decisions have already been made. They often feel that they're forced to live within the Procrustean bed of previous decisions, which often formulate the terms of technical debt and things that weren't resolved by the original design team. So from a process point of view, helping people to see what is fixed and what is variable is very difficult. Because they often times feel that they have to work very narrowly within a range that they didn't decide. They didn't decide the programming language. They didn't decide the operating system version. They didn't decide the hardware. Yet they have to adapt to perhaps new requirements or bug fixes, or optimizations, or other types of enhancements within those constraints.

In contrast, when people do development, they often have much more leeway at deciding what those decisions will be, deciding what the architecture will be, what the tools will be, what the language is and so on. I think that leads to a very different mindset and it also leads to perhaps more creativity in the process environment. You are not necessarily trying to force it, square pegs through round holes and vice versa. Instead you are giving people more opportunity to decide how to work together with the original people defining the requirements and deciding what the system should really be. By the time you get sustainment, those constraints are often much more laid down in stone. This often leads to different teams who are, in a sense if you will, with learned helplessness. They are not accustomed to thinking outside the box. They are accustomed to thinking within very rigid constraints. Therefore, it just leads to a different mentality that makes it hard for them later to transition to a more development-centric role, where that creativity is paramount, especially in the early phases of a project.

Eileen: We have talked about process. We have talked about technology challenges, and we've talked about some of the different mindsets or roles associated with software first sustainment and software development. What are some of the biggest and most pressing people challenges that you have seen? I'll start with Tom.

Tom: Sure. So, during this whole process there has been a slight mantra, right? People do what they're incentivized to do. In sustainment, the rapid repair of defects is what is incentivized. You have people who are on the hook to make somebody's life better with a piece software as soon as possible. There is an incentive there for the rapid resolution of these problems, and that is what the incentive is. In development the incentive is often that we don't know how to do something, and so there needs to be a certain amount of effort put into figuring that out. People in sustainment may become very frustrated with that perspective. Because they are use to just doing. Their entire job is do, do, do, do, and they have a lot of this infrastructure already there; they don't need to know how everything works. That is a little bit countered to what happens in

engineering. In software development, engineers need to know design concepts and patterns. They need to learn to appreciate the non-functional attributes of a system. You know like reasonability, maintainability, or separation of concerns. Those aren't often prioritized in a sustainment setting. So level setting and changing that mindset are paramount for moving somebody from sustainment into software engineering.

Eileen: Thanks. Doug, would you like to add anything to that?

Doug: Sure. One of the common misperceptions people often have about software developers and software engineers is that they are all plug compatible. One is as good as the other. One of the things we discover as we get further into this discipline, is that there are different people with different skill sets and different roles. Just like with everything else, if you go into electronics, you find that the people who are great RF [radio-frequency] engineers, or people who are great at making A6, or FPGAs, and those skill sets are not necessarily infinitely transmissible and beautiful. So the same thing is true with software development and software sustainment. There are people who are really good at testing. You don't necessarily want those people to be your architects. The people who are really good at coming up with detail designs or people who are very good at doing the interfaces of a system, you don't necessarily want them to be doing the implementations. Knowing what skill sets you have and knowing what skill sets are necessary in order to meet the requirements, is really an issue of management. In my experience a lot of times people who were doing software management or software leadership or leadership of software organizations, but they don't have much software background, it's very difficult for them to figure what kinds of roles they need and it's also difficult to assess what kinds of skill sets they have with the engineers that are at their disposal to meet these needs. Trying to get all those different pieces to align in some coherent way is very tricky. So it should be no surprise that when you start taking people who been kind of trained and honed to do sustainment style tasks, as Tom talks about, and then expect them to do a brand new set of things. It might not be a surprise they would have difficulty doing that at first, especially if they don't have a budget for mentoring. They don't have a budget for training. They don't have a budget rebuilding and reskilling themselves to do these new types of tasks. One thing that is also very important to realize is that the sustainment world is heavily driven by cost reduction unlike development and building systems from scratch, which can often overrun monumentally as we all know, by looking at large software fiascos. When you get to the sustainment the budgets are much more fixed and the opportunities to train people and to get them opportunities to learn new skills and new sources of expertise are often very, very limited. They often don't access for example to a research facility the way that you would if you were at a place like Boeing or Lockheed Martin where they have researchers who can help them keep abreast of the latest and greatest technologies and software advances. So sustainment organizations are often times less full of resources to continually retrain and bring their staff up to speed with the latest and great

technologies. Alot of the challenge there is making sure you have the right people for the right job.

Eileen: People of course are not fungible as I think we are all fond of saying. Every development or sustainment organization has adapted its own way of doing things, its own set of local processes suitable to the environment and the system that they have been working on. The technology available and the enabling technologies for different systems can be dramatically different, which also goes back to our people fungibility problem. And we are doing all of this in acertainly a physically constrained environment. These are hard problems to solve. What top strategies would you recommend that organizations emphasize to address some of these critical challenges? I'll start with you Doug.

Doug: First and foremost, it is very important for organizations to realize that core competency in software and software engineering is fundamental to their success. This is becoming more clear nowadays because we see the consequences of ignoring that. But many organizations still have a very hardware and traditional platform-centric view. Software is typically perceived as that unnecessary evil, but it's not really something that they want to be good at. So part of the challenge is to make sure people build a culture, where software skills and software talents are recognized and rewarded. This really comes through with respect to what kind of investment you make in your people. Do you try to hire the cheapest people you can and then just not give them any training or any education to improve themselves? That is certainly one model. But the problem is that those people are very challenged to keep up with the new threats and the new capabilities that are coming online and the new requirements you see in these modern mission systems in the DoD and elsewhere.

I think my first thing is for people to recognize that software is absolutely crucial to their line of business and to treat it same way that they would treat all the other things that they take seriously. To invest in it, to grow it, to build the competencies and to keep abreast of what's going on in the broader world because a lot of what's happening nowadays is really taking place in the commercial sector. The DoD, unlike 20 or 30 years ago, isn't necessarily where all the cutting-edge software is being done.

We can learn awful lot of things by paying attention to stuff like test drive development and continuous integration/continuous development. These are concepts that are very widely applied in the commercial world, and they are only slowly finding their way into a lot of the DoD programs where it is much more expensive to do those kinds of tests without having to go out into some kind of integration lab and spend a lot of money in a hardware and platforms as opposed to simulators and emulators.

Eileen: I am hearing you talk a lot about investing in software as a critical enabler all through the technology and the people and the process through recruiting, through the enabling technologies, through testing. Is that a fair assessment?

Doug: Absolutely, yes. This is a challenge because this is not the way that the DoD has historically viewed their core competency and their value-added dimension. With more and more systems becoming cyber-enabled and all the good and bad that that means in terms of new capability and also new threats, we have to be a lot more intentional and strategic about recognizing the intrinsic value of software as an artifact, and, more importantly, perhaps software as a profession that you want to reward and grow people into.

Eileen: *CrossTalk, the Journal of Defense Software Engineering* recently had an entire special issue dedicated to software engineering as a profession. I will encourage our readers to take a look at that as well. We will include links to that in the transcript. Tom, would you like to talk about core enabling strategies or strategies that you recommend organizations explore and emphasize?

Tom: Absolutely. Core strategies for me, one of the bigger ones, is that if you want these teams and these folks to be able to transition among platforms and software technologies, there needs to be a system in place for that. The simplest one, in my opinion, is having experienced domain mentors for developing the people. Technology is there. There is lots of great stuff out there to pull in and incorporate, but it needs to be evaluated for form and fit. Don't expect to just pull in new technologies because they are cool, but make sure they fit well into your effort. That is where a good mentor or leadership can help with that.

I am big into the whole mentoring idea. So making sure that you have people that have that experience, that may create new people that have that experience. I don't want to say protect them, but I will say that software in at least this industry, in this space, has had a certain utility rather than sexiness to it right? Everything I've worked on in software is a bit of an afterthought. The cool stuff is this hardware, this new technology, but putting it all together, *Oh that's just magic*. So, developing that understanding, as Doug has mentioned, that software is key and critical and is not an afterthought. It is a lesson that has to be more holistically embraced.

Eileen: I appreciate those insights. We have talked a lot about mentoring in particular, in some of our other podcasts in this series and I always appreciate when the conversation comes back to that as we, you know contribute to growing the, growing the workforce that is doing this important work for the long haul.

One aspect of our work that we like to highlight in the podcasts is technology transition. If I am on a team or if I'm managing a team that has recently been tasked to expand emission to include

these engineering and development activities, *Where should I start? What good resources are available to me?* I'll ask Tom to take the first pass at that.

Tom: Oh wow. My general response to that is that you know transition takes time. Patience is the most important resource in that sort of setting. Level setting and adjusting the priorities and incentives is important, and that must be understood by the management of those organizations. In terms of specific resources, I am not super knowledgeable of specific resources. But I will say that focusing on the resources that talk about certain key elements, like the ones that value landing and decomposition before an effort starts. Ones that emphasize the non-functional attributes of an effort are important. There are a lot of efforts for which the expectation is that you can turn on a dime, adjust, and produce a newly engineered software system in three months. Any resources that you look for, that are about transitioning, need to have a much more broad perspective on that and plan for the time it takes for that sort of transition. I always like resources that focus on flexibility and self-organization in their process development and in their people development.

Eileen: Thanks, I appreciate that. Doug, do you want to weigh in on resources that you think managers who are making this transition should be looking towards the first step?

Doug: Sure, you bet. First, I think it's important to recognize one of the challenges of dealing with transition in the DoD historically has been that—because of the requirements, because of the fact that these systems were often designed to be sustained for decades as opposed to just a short amount of time—people are often forced to use older tools, which are not as well supported with the kinds of things we come to expect if you are building an enterprise web system today. Most of the commercial practice, most of the tools from the major vendors like Microsoft and Google and Apple and so on, are really focused on enterprise, e-commerce style systems and web-based systems. Whereas very little historically and what was going on in the DoD was done like that, especially when you are sustaining airport, airplane platforms, or ships, or tanks, or other kinds of weapons systems. So, part of the challenge is to figure out what generation of technologies do people need to be aware of, and then how can they get the experience necessary to be successful.

One obvious resource I'll point people to—because it is one of the few places that focuses almost exclusively on this—is our SEI Blog where we have articles over the past decade, since 2011, that focus on the kinds of technologies, methods, tools, platforms and so on, that are necessary to be successful in the DoD. There are very few other resources out there that are publicly available, that go into that amount of detail, so that is a wonderful resource to start with. It is also important I think to get people to learn modern quality-assurance techniques. Even if you are stuck using older languages and older platforms and older operating systems and so on, you are

still often well placed to success if you can leverage modern quality assurance and testing techniques.

Again, things like test driven development, continuous integration, continuous deployment, those kinds of tools can be applied to almost any environment, and you will be much better off even if you are stuck writing your code or sustaining your code in Jovial or Fortran or lower-level languages like C. Because those techniques and those environments are really very well-versed making people successful at catching a lot of bugs by leveraging. One of the interesting implications of Moore's Law, which is we have more and more transistors on a chip so we can run the computers and we can run the test environments readily 24/7 and do a really good job of automation of the quality assurance process.

Eileen: My understanding is that you are continuing to develop the blog post series, that there are two of them now [process], [people] and that there are more coming. I think they are both a really great read. They reference some really important work, and they make the concepts really easy to sort of sit down and break down and recognize challenges that you are encountering as you are making changes in the organization. I am looking forward to seeing what is next in that line of work.

Doug and Tom, I want to thank you for talking to us today. I've really enjoyed it; I hope you enjoyed it as well. For our audience, we will include links in the transcript all of the resources mentioned during today's podcast and thanks again for joining us.

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.