Reliability Validation and Improvement Framework

featuring Peter Feiler interviewed by Bill Pollak

Bill Pollak: Welcome to the SEI podcast series, a production of the Carnegie Mellon Software Engineering Institute. The SEI is a federally funded research and development center at Carnegie Mellon University in Pittsburgh, Pennsylvania. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts. My name is Bill Pollak, and today I am pleased to introduce to you Peter Feiler, a senior researcher at the SEI. Peter is the technical lead and author of the Architecture Analysis and Design Language Standard, which was published in November, 2004. Version 2 of the standard was published in January, 2009. Peter is a senior member of the technical staff, and his research interests include dependable real-time systems, architecture languages for embedded systems, and predictable system analysis and engineering. In today's episode, we will be discussing the findings in an SEI technical report that Peter co-authored on the Reliability Validation and Improvement Framework. Peter, thank you for joining us today.

Peter Feiler: You're welcome.

Bill: So to borrow a question from Heilmeier's Catechism, what is the problem that your research addresses?

Peter: Well, the problem is the following: We have safety-critical systems such aircraft, and they are more and more driven by software that provides the functionality. The problem is—when we try to qualify or certify those aircraft—that has become more and more of a challenge. Organizations that are signing off the papers to certify or qualify such an aircraft have a harder time having confidence in the fact that the system is safe.

Bill: So, certification and qualification are primarily not safety.

Peter: That's correct. One of the things that we are talking about here is something that—in this report we are looking at it from a qualification/certification perspective, but the underlying technologies are also valuable to the people building those systems. That's what this framework is addressing.

Bill: I see. So, please give us some background on reliability engineering and how it applies to your research?

Peter: Reliability engineering has its roots with people building physical systems, like the aircraft itself. The focus of that has been [on] physical parts [which] have mechanical failures. So, what you're doing is, based on experience, you have a sense of how long does something last before it breaks. That's what reliability is based on. When we do that for software, researchers have had a hard time coming up with good figures. The reason is because software errors are design errors, not errors in the physical system.

For physical system engineering, usually things evolve very slowly. Aircraft—the structure of aircraft—hasn't changed in the last however many years. Or, when they come out with a new type of engine, yes at first they have design errors. But, after a while, it is mostly focused on the reliability. So, we have to change our attitude about how we look at reliability, because we cannot really assume that software has zero defects. So, there are design errors in software, but we have to tackle it in a new way.

Bill: Did you actually develop a Reliability Validation and Improvement Framework? Please give us an overview of how you developed it.

Peter: Yes. The way we got into this particular piece of work is actually an organization on the DOD side. The Aviation Engineering Directorate in the U.S. Army is an agency that has to sign off on the airworthiness of rotorcraft. They said, "We need a new way of looking at the problem space and the strategy for addressing this." So, they came to us at the SEI to address it.

When you look at the problem, what we have done is identified four areas that are contributors to making a system that is heavily reliant on software higher quality. The four elements are—first of all you want to have your requirements specified in the better form, because when you have requirements ill-specified, you have already introduced 35 percent of all system-level errors.

The second piece is that, mainly due to architectural design, we introduce another 35 percent of errors. So, focusing on those two errors and supporting them by having a way of more formally specifying them, we can validate the requirements. Then, we can verify the initial architectural design against requirements. One of the challenges is how to do that for requirements that are not just basic functionality but what we call non-functional properties or quality attributes. That is things like reliability, safety, performance, and so on.

It turns out, in that area, recently there has been an interesting piece of work done for the FAA (Federal Aviation Administration). A group of people from Rockwell Collins who are heavily into model-checking type of technologies did a study on our current practices and requirements in engineering, and then wrote a handbook that includes an 11-step process that leads you towards specification of requirements in a more systematic way. In a way that then lets you do some analysis on it by having, for example, take them to representations of state machines and things like that.

Bill: I see.

Peter: So, the second pillar focuses on the architecture itself. This is where we come in with architecture modeling. One example of a notation that helps in that area, obviously, is the SAE AADL, which was designed specifically for embedded systems.

The idea in that area is, "How can we—through virtual integration, based on these architectural representations and detailed design representations—find out about problems that have to do with system integration much earlier in the process?" That leads to some of the work that is going on in the aviation industry under this notion of SAVI or System Architectural Virtual Integration that Boeing Airbus and Embraer and a whole bunch of companies are doing. I think we had another blog or podcast on that topic. So, that's my second pillar.

The third pillar is to say, given these representations—whether it's the requirement or the architectural representation—can we use static analysis, namely analysis of your design to identify potential problem areas? So, it's predictive analysis. We still have the problem that the model we use is not reflecting the real system implementation. So, we need to keep that alive, but we can eliminate problems that otherwise aren't detected until system integration.

The current practice is design a system, build a system. Then, you put it together. Then, you try to determine whether it works, and that's rather late. So, we want to change it around to say, "Let's virtually integrate first. Make sure it fits together, and then build pieces against it.

Then, the final piece is, how can we have confidence in the work? That's where we pull in the assurance case type of work, building a case for, "Why do I have confidence in this thing?" by keeping a record of all the different forms of evidence that we are building up throughout the whole development process.

What that then does is—it leads us to a development framework and also qualification framework that go hand-in-hand in parallel. We have this V-model of development where on the left-hand side of the V, we have requirements, design down to code. On the right-hand side we "walk out," unit test, and all that stuff. Whether you follow that in the waterfall fashion or spiral or whatever, it doesn't make a difference. But that's the basic framework. What we are now saying is, instead of having the collection of evidence only on the right-hand side, basically make it a double V, where during the requirements engineering, architectural design, we are already collecting evidence, namely by doing this analysis of the representations at that point already.

Through all of those we then reduce the kinds of errors that are leaking design errors that stay behind and leak all the way through into system operation. That then lets us improve the quality of the system and, based on that, we can have then some measurable improvement on those things.

Bill: So, were there any surprises in the research? And if so, please tell us about them. If not, were there any challenges that you could tell us about?

Peter: Well, the surprises in some respects or challenges are that it's a multi-dimensional problem. What I mean with that is that we can focus on the system itself. On one hand, "What are areas in the system that can cause problems?" For that, we actually had some research that specifically looks at "What are software-induced errors that are brought into the system because of the things we do in software?" There's a whole set—a report and a topic on that whole thing. But, we also need to consider the development process. "What kinds of things can go wrong there?"

One example of that is if you have a reliability model done by hand, a mark-up model where you predict a reliability and then you do work-load balancing, for example, across process areas later on, so you move software around. You may actually lose some of the reliability that you had assumed early on in the other model. So, your models are really inconsistent with each other, so the model results have little meaning.

At the same time, we made assumptions in the development process. They are usually not documented. If you have a record of those, when then later on in the process we make some adjustments in the architecture—like in this case just on the deployment—we were able to say, "Hey, you made some assumption here, now you're violating those assumptions." What that lets you do is take a new look at the way you go about verifying and testing systems.

So, if it's testing (we are not trying to do away with testing), so all your flight tests and all those are in place. What we are trying to do is two things: One is, to say, well, "Certain tests will pass the first time around, because we found the problem much earlier."

Bill: So, that saves development time.

Peter: That is then the rework-avoidance kind of thing. We save development time. The second thing we do is to then say, "If we understand what the assumptions were that we were making up front in our analytical approach, can we then design some tests specifically for those assumptions?" One simple example of that is when I do scheduling analysis, one of the inputs is, "What is my worst-case execution time? What's the worst case? How long does it take for this piece of software to do its job?"

Well, we put some numbers in there. Once you have a running system, I can now design a test that actually measures and tests for "What is that?" If, in the actual running system, that number is higher than the original analysis, obviously then the original analysis results aren't valid anymore. But, we can just feed those revised numbers back into the model and then revalidate even analytically to complement the traditional testing.

This hand-in-hand that I think is a real interesting approach to simply say, "It's not an either/or, but what we want to do is for both sides tackle the problem."

Bill: Great. So, what's next for your work in this field? What's the future direction of the research?

Peter: Well, this is just a framework that lays out the whole problem space. What we have is, in several of these areas, we have the actual technology pieces working. Even at the SEI we have work in assurance case-type of area. We have work in model checking and schedulability. We have work in architecture-centric modeling. We are doing some work, not necessarily that strong at this point, but we're building up around how to capture requirements. What matters now is making those four pieces interplay in a systematic way.

That's now happening both within the SEI as well as initiatives like the SAVI initiative that I mentioned before. They are currently in a phase where they're actually going through an exercise of modeling an aircraft landing-and-breaking system from the beginning through the end. Touching on requirements, architecture analysis—all the way through the tail end of it to build up a safety case, following practices as they have to document it.

One of the things that fascinated me is this is an area where they have lots of safety practices in place. Why is it that we still have problems? The issue is, like we said at the beginning, it's not so much that the practices aren't good, but in those practices we have not fully understood what are the contributions from the software, or what I call software-induced faults, that affect the system as a whole.

That piece of understanding, that is the new research contribution in that sense, that we are trying to get into practitioners hands, because we have some reasonable insight on that and are trying to make practice better.

Bill: Great. Peter, thank you for joining us today. To download the technical report that Peter co-authored on this topic, *The Reliability Validation and Improvement Framework,* or any of the SEI technical reports and notes, please go to sei.cmu.edu/library/reportspapers.cfm. This podcast is available on the SEI website at sei.cmu.edu/podcasts and on Carnegie Mellon University's iTunes U site. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you.