Predicting Software Assurance Using Quality and Reliability Measures

featuring Carol Woody and Bill Nichols as interviewed by Suzanne Miller

----

Suzanne Miller: Security vulnerabilities are defects that enable an external party to compromise a system. Our research indicates that improving software quality by reducing the number of defects, also reduces the number of vulnerabilities, and improves security. In this podcast we will discuss how a quality emphasis in software development can improve software security. 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 sponsored by the U.S. Department of Defense, and operated by Carnegie Mellon University. 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, I am very pleased to introduce you to Bill Nichols and Carol Woody, who we have both spoken with before on other topics. In today's podcast, we are going to be discussing their research in using quality and reliability measures to predict software assurance.

First a little bit about our guests. Carol Woody has been a senior member of the technical staff since 2001, and is the technical manager of the Cyber Security Engineering Team, whose research focuses on security and software assurance for highly complex networked systems throughout the development and acquisition lifecycles. I've worked on several projects with her on different aspects of this, and it is continuing to provide lots of challenges as we go through the topics.

Bill Nichols joined the SEI in 2006 and originally served as a [Personal Software Process] PSP instructor, and TSP coach with the Team Software Process program. Bill's current focus is on performance data and data-driven analysis of the software development process. Prior to joining the SEI, Bill led a software development team at the Bettis Laboratory near Pittsburgh where he had been developing and maintaining nuclear engineering and scientific software for 14 years. Welcome to both of you. Thank you for joining us today.

Carol Woody: Our pleasure.

Suzanne: Let’s start off by having you explain to our audience what you mean when you talk about software assurance. We hear that word a lot, and there are some different definitions of it. So what does it mean in the context that you’re working in right now? Why is this such a difficult topic?

Carol: Software assurance is always spoken of as software that functions as intended and is free of vulnerabilities. Unfortunately, all software has defects, which indicates that there are problems with it functioning as intended at different points in time. We have yet to really be able to prove that any software is totally free of vulnerabilities. So, that puts us in a situation of knowing that there are problems, but basically we want to reduce them as much as possible. In our context, what we view is that all vulnerabilities are essentially defects of one form or another.

Suzanne: They are security defects is one way to look at it.

Carol: They are security defects, but they are still under the large quality umbrella as defects. So, that would suggest that there is real value in looking at what quality provides in terms of the approaches to addressing this problem space.

Suzanne: So, basically the question you are asking is, *can some of the techniques that we use for addressing defects in general for software quality, can those be applied to actually improve security defect detection and management?*

Carol: Exactly. Bill, you may want to talk about the data we found.

Bill Nichols: One of the things I will point out here is that injecting defects is really kind of an inevitable byproduct of software development because it is done by humans. Humans make defects. So, if you build a product, you are building defects at the same time.

Removing them is another matter. There is nothing that forces you to remove all those defects, and it is pretty hard work. That is why software assurance has been so difficult, because you have to apply techniques with a lot of diligence that identify these defects and remove them. Now, what we have found from the research was that a lot of the security vulnerabilities were injected during the implementation phase. For all of the design considerations that you would put into your development, these could be easily undermined by fairly trivial types of defects. What we were finding was that some fairly large portions still of the security vulnerabilities should be removed if you remove the common defects. That is, you have this big dragnet; remove all of these defects; you are going to get a pretty fair number of security vulnerabilities along the way.

Suzanne: There are software applications that help with this. There are code analysis kinds of tools that will identify certain kinds of common potential defect areas. They aren't always defects, but they identify some of these for you. And then you have choices about how to fix them and things like that. Are you able to use some of that kind of code analysis application for this? Or, is this a completely manual process at this point for the security vulnerabilities piece?

Bill: Well, you ask a very interesting question. What we found is that there is no single silver bullet. You have to fire a lot of lead bullets. Each of them works sort of effectively, but you need to fire a lot of them. So, you apply a lot of your standard, quality techniques. You do good design. You go, and you review the designs. After you have done the coding, you have peer inspections of the code. Then you can start applying some of these other tools with static analysis. If you haven't been applying the quality assurance, what we were finding is that you would be overwhelmed with the responses from a lot of your static tools. You are inundated with so many messages, you don't know what to do next. But, the organizations that were applying strong quality assurance techniques were getting substantially fewer warnings. And, it was much easier for them to manage them.

Suzanne: So, some of this is about teaching the world, and particularly the implementers of these kinds of software—well, any kind of software actually—what are the common failure patterns in both design and implementation that would lead you to these kinds of vulnerabilities, so that they can avoid those programming practices and design practices. You want to couple that then with static analysis tools that hopefully become more informed by what your research shows are the bad patterns in implementation. Is that a good way of saying what you need to do?

Carol: In some sense. But, when you think about software functioning as intended, security breaches usually come from someone taking advantage of the fact that the software doesn't function as intended. Simplistic quality problems such as buffer overflows are still extremely common, and yet your standard quality techniques have been focusing on attempting to remove those and have procedures and processes for tracking those. So, if we look at some of the especially low-hanging fruit of high security problems, like the top 25 that get bandied around in many conversations, many of those are quality problems as well as security problems. So, by tackling the problem jointly, we would view this as a real opportunity to leverage the knowledge and capability of both of these areas to really make a difference and improve software assurance.

Suzanne: So, there is this development side, which is looking at preventing, if you will, these from getting into the field. There is also the operational side, which is looking at how *do you, essentially in the maintenance phase, avoid re-injecting additional defects or defects that had been removed. Are you dealing with both sides of that?*

Carol: We have looked primarily at that the development side, but there is no reason to restrict the areas that we have been looking at. The techniques, the approaches are certainly applicable.

Suzanne: Because one of the things that we find in other research in sustainment is that you don't always get communication across that development operations boundary as to the kinds of things that were explicitly avoided or explicitly done to meet some kind of quality attribute. So, the sustainers, if they don't know that these things were done to protect security, they may undo some of the good implementation by using a practice that would be considered something that would add a vulnerability.

Carol: My guess would be—and this is based on my experience working in sustainment, which I've done for many years—is that you are more likely to see a situation where someone is rushing to put in the fix, because it is a crisis situation, and they not really doing the good, quality review checks and analysis practices.

Suzanne: Even if they know about them.

Carol: …even though they know about them. So, that tends to be where we get the re-injection of problems on the sustainment side. Quality approaches would help, but they do tend to be more time consuming. So, you are playing with the trade-offs of how well do we verify things before we implement to make sure that we haven't done something wrong. I think that boils down to how important is it that the software function as intended. If there is a high risk of problems there, related to either quality or security, then a serious look at how the work is being done should be taken.

Suzanne: OK. Bill, you had something to add?

Bill: Yes. One of the things that we have observed is there really isn't a fundamental difference between developing new software and doing maintenance.

Suzanne: But, it is often different people.

Bill: It is often different people, and that can be the problem. But, the same techniques generally apply. I mean one could easily joke, without being too far off the mark, that there's only been one program ever written, Hello World, and everything else has been maintenance on Hello World.

Suzanne: We've all been there.

Bill: But, to your point about the injections, in the current environment, some of those common defects are still dominating the problems. An example would be one of the Mars probes—it was an orbiter—, which was destroyed by a software update. It injected a buffer overflow. So, it was the maintenance, the bug fixes, that actually destroyed it. If you look at some of the industry data, Capers Jones says about 7 percent of all defect fixes are bad fixes. Our data says it is about twice that, but you find about half of them in test. Think about that for a minute. Every time you fix a bug, you have got a maybe a 1 in 7 chance of injecting a new one, and about half the time it is going to escape your testing.

Suzanne: Which, depending on the software characteristics could be pretty devastating.

Bill: Well, one of the things we found in our research was somewhere on the order of 1 percent, maybe as many as 5 percent of all of the defects can be exploited or associated with vulnerabilities. Let's put it that way. So, there is another way you can quantify this.

Suzanne: So, you are building awareness through the report that you have recently provided of some of these vulnerabilities and how they connect to common software quality defect patterns. I'll call them that. One audience is programmers. Another audience is software testers, so that they can do a better job of testing for these things. Do you see the education community getting involved in this? Because, to my mind, that is where everyone learns how to do Hello World and everything that comes later. Are you seeing interest from the education community in understanding these kinds of vulnerabilities and software quality connections more deeply?

Carol: There is an extensive effort for improving software assurance within the educational community, primarily because so many students graduate with knowledge of how to program, but not really knowing the impact of potential problems that can occur. I am sure that other podcasts will talk about the curriculum model, and the software assurance competencies that have been defined and are being rolled out into the university levels. I think another community that you haven't touched on that may want to consider this research is the acquisition community.

Suzanne: Good point.

Carol: Because there is information about defects. So, that would say that if you know what the quality of the code is, then this 1 to 5 percent gives you a way to think about what kind of security vulnerabilities you may be adopting when you are implementing the software.

On the flip side, if you look at the vulnerability databases that are being tracked, and that information being assembled—if you don't know the quality of the code, but you do have a laundry list of vulnerabilities—then you can start to make some projections about the quality of the code. If you assume that you are really going to know 1 to 5 percent of the information that you should. Bill, I think you were working with Heartbleed on some of that information to see what you could learn about it.

Bill: Exactly. I looked at Heartbleed. There was another go-to-fail; a similar type of problem with some Apple code; very conventional types of defects injected during maintenance, or that had been there for a while.

If you go back at some of the data, Heartbleed was an open source. You could see the number of vulnerabilities found over a period of years. You could make an estimate that having another one is really no surprise. It's more of the same. So, you look at your historic data. You can see that there has been a trend. We have had a certain number of defects that have been found, a certain number of vulnerabilities. Well, nothing is going to change until you change something more fundamentally. I think another thing I would look at is, all of the defects we find in the code are really sort of the hay—that stack of hay—that is obscuring searching for the needle of the vulnerabilities. If you don't have the good quality code, you just get swamped by the types of warnings.

So, I think the message is all of those things that we know about, specific vulnerabilities and design techniques, are important, but to make them effective, we have to apply real, serious quality assurance techniques first.

Suzanne: And, quality design. Not just assurance.

Bill: Quality design and assurance. They all go together.

Suzanne: Right. Right. Never want to forget that they all go together.

Carol: Full lifecycle. We agree.

Bill: Exactly.

Suzanne: OK. So there's a lot here. There's things for acquisition people to pay attention to. Developers to pay attention to. Software quality assurance professionals. Testers. Sustainers. All sort of have a part in this. How can they really make use of this information to really attack this aspect of quality and vulnerability affecting each other? What kinds of recommendations have you made?

Carol: I would reemphasize, I think, what Bill mentioned earlier. That is, you really have to look at the quality practices you are implementing. It focuses around *what are you doing to remove defects as you are going through?* Not just running tools and reacting to them because the tools won't find everything, and the tools may not find the key problems. But, the combination of good quality practices and a focus on defect removal as well as the vulnerability tracking tools gives you the best approach that we've seen.

We have seen five or six specific cases where this strategy has really produced outstanding results.

Suzanne: Good. Good. Good. We look forward to continuing with this and to things like, getting this into the education system as well as into the acquisition system.

Thank you very much for both of you joining us today. To view the technical report that Carol and Bill have authored, which provides a deeper dive into this research, please visit the SEI digital library at resources. sei.cmu.edu/library. In the search field, enter the name of their technical note, which is *Predicting Software Assurance Using Quality and Reliability Measures*, to view all related resources on this topic.

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 for listening.