Common Testing Problems: Pitfalls to Prevent and Mitigate

featuring Don Firesmith interviewed by Suzanne Miller

Suzanne Miller: 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. I am Suzanne Miller, a principal researcher here at the SEI, and today I'm very pleased to introduce you to one of my friends and colleagues, Don Firesmith. Don's work focuses on testing requirements, engineering, system architecting, and product lines. He's kind of a jack-of-all-trades. Aren't you, Don?

Don Firesmith: Yes, a bit.

Suzanne Miller: This most recent work that you've been doing is related to categorizing software pitfalls and testing in particular.

Don Firesmith: Right.

Suzanne Miller: It's not the product-line requirements. This is focused on the testing side of things. And the data on the cost of inadequate software testing and software-system testing it's staggering. It's in the billions of dollars as you've documented it. A lot of this cost is due to rework based on the fact that testing isn't always the best method for finding defects in software code. You cited a study by Capers Jones that less than half the defects in code are typically found by testing. So, what does that mean for the systems that we rely on that are filled with software? This is more than just a testing problem.

Don: Right. There's basically two answers to that. Obviously, we need to do testing better. It's very frustrating to see the same problems occur over and over again on programs. We've got to learn the lessons from testing and apply them on future testing programs.

Testing by itself just isn't going to get the job done. Testing typically only finds maybe 50 percent of the problems in the code. Since a lot of the problems are introduced during requirements engineering and architecting, it really makes sense to try to both prevent those problems up front and to find the problems then instead of during the typical test cycle when they're much, much more expensive to fix.

In fact, they can cost anywhere from five- to a thousand-times as much to fix during unit testing through system testing as they do if you'll fix them during requirements engineering and architecting.

Suzanne: Isn't there some other SEI work that addresses some of these larger areas of system reliability, especially in the requirements and architecting phases of software development?

Don: There actually is. Peter Feiler did a blog entry very recently on some work we've been doing. It's essentially a framework for improving and verifying reliability, safety and security. One of the nice things it has done is it has allowed us to identify these problems and fix them much earlier.

We've essentially got a four-part approach to solving this problem. The first part is getting the requirements down in an analyzable form. The second thing is to take these requirements, build an architecture—architecture models that are themselves analyzable so that you can determine whether or not the architecture supports the requirements. The next thing to do is to use static analysis on these models to find the defects upfront during requirements, engineering and architecting.

Suzanne: And, to find some of the defects that you can't find through testing code, right?

Donald: Exactly. Testing only finds certain kinds of defects. We talk about verification as a four-part thing—testing certainly but also inspections, reviews, and demonstrations. There's actually probably 10 or 15 different ways of verifying requirements being met. Testing is extremely important, but it's only one way.

Suzanne: Right and I distracted you from the fourth part of this approach that Peter's using.

Donald: The other thing is we basically want more confidence early on that our systems are going to be reliable, safe, and secure. By using assurance cases, which are based on the idea of safety cases, we can develop all the evidence that we need to show that the system has the properties it needs to have.

Suzanne: That's one of the simplest, coolest technologies that I think the SEI has worked on in the last 20 years. It's one of the really underutilized [technologies].We'll do that in another podcast, but assurance cases are wonderful.

Donald: **A**nyway, read Peter's blog.

Suzanne: If you're interested in Peter's blog, which has been referred to by Don, please go to our SEI blog site, and you'll be able to click on it from there. So, Don, what is your goal with this research to classify and categorize common testing problem pitfalls.

Donald: My motivation was primarily due to frustration. I do a lot of independent technical assessments, looking at a lot of testing processes and programs. I keep seeing the same problems over and over again. I wanted to collect this information together, so that we can give it to the test organizations so that they'd know what to avoid in the future. The other thing I wanted to do is I wanted to collect the information about all of the commonly occurring test pitfalls.

Suzanne: There's a lot more than you would think.

Donald: There is a lot more than you would think. Most of the papers and presentations that talk about testing pitfalls have a handful, maybe as many as 10 pitfalls. By looking at all the different programs that we've looked at in testing, I found 80 different things that show up over and over again.

Suzanne: That's frightening.

Donald: It is.

Suzanne: I've had some of the same experience, and I understand the importance of what you're doing.

Donald: People are very creative on how to not do testing right.

Suzanne: That's true. So, to to get your arms around this, I saw that you've binned these testing pitfalls into two major sections: general things (a lot of those are process-related communication, planning, those kinds of things) and some that are test-type specific (unit testing, or domain, or low testing, those kinds of things). Which of those categories is the easiest, the least costly to mitigate, or have you had a chance to even deal with that part of the problem?

Donald: I think you have to look at it in more detail than just the general testing problems and the test-type specific problems like unit testing or regression testing. In fact, I haven't been able to really deal with that yet.

Right now, my first goal was to identify the problems, figure out what the symptoms were to tell you whether or not you were suffering from these problems. What were the negative consequences if you did fall into this pitfall? And, recommendations on what to do about it—either to prepare, enable yourself to avoid the problems, or to dig your way out of the pitfalls if you find yourself in them.

In the future what I want to do is I want to find out which of these problems show up the most often and which of the problems have the biggest negative consequences, the problems with the highest risk. Since there are 80 different types of pitfalls that we've identified, and no one can really deal with all 80 at once, we'd like to prioritize them to see which ones create the most

problems. I hope in the future to do a survey with industry to find out, just in the field, what the differences are.

Suzanne: So a sort of Pareto analysis where you consider which 20 percent gives us 80 percent of the problem kind of idea.

Donald: Exactly. While it is important to at least be aware of all of them, if we're going to start avoiding them, there are some that we need to concentrate on first.

Suzanne: Now, one of the things that impacts this is that software development methods are changing. We're seeing a whole class of methods that are generally termed "lean" or "agile-software-methods." Those methods drive you towards continuous integration, daily builds, automated testing that's very robust, lots of regression testing. Do you think that will have any effect on the prevalence of some of these test pitfall categories?

Donald: It definitely does. It does help with some of the pitfalls to avoid them or to keep them down. On the other hand, every approach tends to have its strengths and its weaknesses. For example, if you're doing an agile program, you tend to have a lot of requirements churn, which in general, a good thing because you want to get the right requirements. But, since you're constantly changing the requirements, it makes it much harder for the test people to keep up with this, because as they develop some of the test cases they have to change with the requirements.

Suzanne: Right, and they're not quite as accustomed to that level of churn. They're accustomed to doing a test plan that's much more static.

Donald: Right. The other issue here—and you mentioned it—is with the regression testing. If we have an iterative approach, we're constantly fixing defects. We're constantly changing the code, which means that we constantly have to perform regression testing. That's only going to be practical if we automate the regression testing. On a lot of programs that do an agile approach, they tend to be fairly good at automating unit testing. But, as you start getting up to integration testing and system testing, it tends to fall off. There are [also] certain types of testing that just can't be automated.

Suzanne: Sure, sure.

Donald: A lot of the things when you're dealing with, for example, testing the usability of a user interface and so on, it's very hard to automate that sort of thing.

Suzanne: We're not testing the user interface for robots. We're testing the user interface for humans, so you're going to have to interact with them.

Donald: Exactly. So, while it does help in certain areas, it does make things more difficult in other areas.

Suzanne: I know you're familiar with one of the gurus of the agile movement, Bob Martin. He's fondly known as "Uncle Bob" to a whole generation of software programmers. He's been quoted, I actually heard him say this in a conference, that "A programmer shouldn't be considered a professional if he or she doesn't use test-driven-development," the technique where you write the test case for your code before you even do the code. That idea is that if you can write the test, you can probably write the code that will pass the test, and things will be better from there. Is that another strategy? Is that another different method that people are using now of programmers becoming better at writing test cases that could solve some of these pitfalls?

Donald: I think it is. Personally, I've been doing this for decades. The idea, when I develop my own code, of trying to figure out what the test cases should be [and] for that matter, what the assertions should be: the preconditions and post conditions. If I understand how the code should work, it is easier to write the right code.

On the other hand, this is another case where it helps when the developer is testing his own software, which is usually at the unit level. It becomes much less practical when we're talking about integration testing and system testing.

Suzanne: Sure. And acceptance testing.

Donald: And acceptance testing, where you have a separate test team of test professionals who are testing the work done by other people.

There's also another bit of a pitfall here as well that you can get into. Certain people, in the agile community especially, have mistaken test cases for requirements. While they're closely related, they are quite different concepts. You only can do a certain number of test cases, and yet your requirements are typically much larger. So, you're selecting appropriate test cases. You can't assume that those test cases are the requirements because you want to do more than just that.

Suzanne: That gets back to the multiple types of verification. You don't want to just rely on testing as your only method for verification, especially for requirements that are not as amenable to testing as some others.

Donald: So, the test is one method for verifying the requirements, but the testing is not a replacement for the requirements.

Suzanne: Right. So, you talked about wanting to do some more analysis of "What's the prevalence of these pitfalls?" "Which ones tend to cause the most problems in terms of system deployment and those kinds of things?" Are there other things that you would like to do with this research if you had the time and funding?

Donald: Right now, we've looked at what's happened up until now. But, the situation is changing. We have new methods, new processes, new test tools, new application domains, and so on. It would be interesting to track this over time to see how things change. I'm fairly certain that [some] of these pitfalls will become more prevalent or more important, higher risk. The question is "Which ones are they?"

The other thing is, as I said, the application domains are changing. While I recently, for example, added pitfalls dealing with mobile devices, there are other issues: autonomous systems, robotics, and so on that are coming down the pike at us. The question is "What are the test pitfalls that are most relevant to them?"

Suzanne: And, we'd like to find those out before the people that want to hack into them do.

Donald: Exactly. The other thing is software, size, and complexity has been going through the roof. It's exponential. We're going to be doing much larger, much more complex systems five years from now than we are today. We can't continue doing things the way they are now. The cost of testing just becomes unaffordable if we keep trying to do testing the same way. So, as things get larger and more complex, what can we do to avoid these pitfalls?

The last thing is, while I've dealt with some system-of-systems pitfalls, it would be interesting to know just what kind of testing pitfalls are most prevalent with product lines.

Suzanne: Okay. Well it sounds like you're going to be busy.

Donald: I hope so.

Suzanne: Well, thank you Don very much for joining us today. If you as a listener want more on Don's work in this testing area, he does have a second blog on this series and from what he's saying I think there will be more coming. We look forward to that.

If you'd like more information about the SEI's recent research in all areas, you can download all of our technical reports and notes at sei.cmu.edu/library/reportspapers.cfm. Listing for papers, blog posts, and podcasts related to SEI research on this and the Agile adoption and DoD work which we referenced earlier can also be found at sei.cmu.edu/acquisition/research.

This podcast is available on the SEI website as I said 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 info@sei.cmu.edu. Thank you.