Four Types of Shift Left Testing

featuring Donald Firesmith as Interviewed by Suzanne Miller

Suzanne Miller: Shift-left testing, which simply means beginning testing as early as practical in the lifecycle, is one of the most important and widely discussed trends within the software testing community. What is less widely known, both inside and outside the testing community, is that testers can employ four fundamentally different approaches to shift testing to the left. In today's Podcast, we will explore these four approaches and which one might be most appropriate for your organization.

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 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 pleased to introduce you to my colleague Don Firesmith, the genius behind these four approaches and a fellow principal researcher here at the SEI.

Today, we are going to talk about Don's research into the four approaches for shifting testing to the left. First, a little bit about Don, who has been a guest on our show previously to talk about his other research in software testing. At the SEI, Don provides practical guidance, for example, with regard to requirements engineering, system and software architectures, and testing to help program offices acquire software-intensive systems.

He also develops and maintains new technologies such as methods for engineering safety and security requirements, as well as accessing the quality of system requirements and architectures. He is also the author of countless journal articles on software and systems engineering, as well as seven technical books including, most recently, *Common System and Software Testing Pitfalls*.

Don, thank you for joining us today. How does shift-left testing differ from traditional testing?

Don Firesmith: Well, traditional testing, and by this I mean testing of 15, 20, 25 years ago, was testing pretty much at the end. You would have a waterfall lifecycle. You do your requirements, your architecture, your design and implementation, and only when you were done with all of that work would you finally get around to doing your testing. Even there—if you look at the right side of the traditional V-model, which starts with unit testing, then integrating testing, then up to systems testing—back in those days, a lot of people would start right off with system-level testing. They wouldn't really do much in the way of unit testing or integration testing.

Suzanne: *We'll catch it at system testing*.

Don: Yes, *We'll catch it in system testing*. Often, this was testing at the very end, which had the problem that if you found anything, you often didn't have time to fix it before your delivery date.

Shift-left testing, on the other hand, is essentially moving testing to the left on the development lifecycle, left on the project timeline. Therefore, what we are doing here is moving testing earlier, so that we are not waiting until the system is produced to start testing it. But, we are starting to do much more in the way of unit testing and integration testing and performing these activities much earlier, beginning to do testing at the very beginning of the project.

Suzanne: We are not just talking about defining the tests at the beginning of the project because that has been done for years, but we are talking about executing tests at the beginning of the project.

Don: That is exactly right. That is a key distinction. In the past, the left side of the V was there for planning and doing some initial preparation work. You might actually develop some test cases early on, but you weren't actually testing anything because, back in those days, there was nothing really to test at that point in time.

Shift-left testing is based on other trends in the software development world, which has moved a lot of the rest of the development effort earlier in the project. We started doing detailed design work earlier. We started doing implementation earlier. We started doing integration earlier and, therefore, we had something that we could start testing much earlier.

Suzanne: Cool. Why does it matter? Why does shift-left testing matter? What are its benefits, and are they are any drawbacks to this approach?

Don: The benefits of shift-left testing include things like the testers become involved earlier in the development process. That makes it more likely that testing is going to get integrated with the rest of the development process, and it is also going to make it more likely that testing receives the resources it needs to get done properly. Testing executable models can be used to identify mistakes in the requirements, the architecture, and the design. This is done then before significant efforts have been wasted in implementing these in software and hardware.

It also provides more time to fix these defects found by testing, thereby decreasing the likelihood that these defects are going to get postponed to later increments or versions of the system. That helps us avoid creating a bow wave of technical debt that sinks projects if it grows too large.

Suzanne: So, these are benefits that are very important. A lot of these have schedule implications, and they have cost implications. Why wouldn't everybody do it this way?

Don: Actually, certain types of shift-left testing are becoming extremely popular, sort of the standard way of doing things now.

Suzanne: Certainly within the Agile community, having testers on the development team from the very beginning is pretty much a standard practice. There are some parts of the community that have really embraced this kind of approach, but, clearly, not everyone has or else we wouldn't be talking about this.

Don: A lot of people, especially on smaller programs, smaller companies, haven't really made the transition yet. One of the types of shift-left testing, the testing of executable requirements and architecture models, is just beginning to be performed now because there is not much in the way of good tools for performing the testing, and a lot of people just aren't aware that it is an option.

Suzanne: Executable models, although they have been used in Europe for many years, are really just becoming a thing in the United States engineering communities. The existence of those models…you have got to train the engineers how to do the models, and you have got to train the testers how to interact with the models. So, there are some barriers to that particular type of testing that exists today, but maybe not forever.

Through your research, you have identified these four approaches to shift-left testing. Why don't you describe each one of them for us briefly.

Don: Traditional shift-left testing is basically moving from the upper right side of the V to the lower right side of the V-model. What we are really talking about is moving the emphasis off of the system-level GUI testing with things like record and playback tools down to greater emphasis on unit testing, integration testing, API [application programming interface]-level testing. The transition to this kind of shift-left testing is largely completed amongst professional testers.

Suzanne: OK. What is the second?

Don: We'll refer to the second type of shift-left testing as incremental shift-left testing. Instead of having a single V-model for a single waterfall lifecycle, what people will often do is realize that a system is going to go through various major increments. For example, if your system is going to go through four increments, you will have four, if you will, V-models.

The testing that occurs on the first two or three of those V-models happens before that testing would have happened had you had just the single waterfall lifecycle. It's a moving to the left by dividing the projects…

Suzanne: By breaking down the system.

Don: …into rather large increments.

Suzanne: OK. And the third?

Don: The third, you already mentioned and that is Agile testing and even more so, testing in DevOps. It is very similar to the second type, but, here, it is a matter of size and duration of the increment. What we will be doing here is doing sort of a continual integration and testing where you can look at it as many small V's instead of much…

Suzanne: So, I talk to people about the incremental model being like a W and then I talk about Agile being a saw tooth, right.

Don: Exactly.

Suzanne: It is smaller increments but lots of increments.

Don: Yes and, in fact, not only do you have lots of increments, so a great number of these increments have their testing happening before they would have even with an incremental approach.

The other thing that often happens on large projects is you will have multiple teams working on different subsystems, different major parts of it. Consequently, that also then tends to move the testing on those increments earlier in the development process.

Suzanne: One of the things we are seeing in Agile is even specialty testing, such as information assurance, is also moving to the left where they are getting more engaged in the individual iterations instead of waiting for releases or waiting for final development.

Don: Exactly. It is not just unit testing and integration testing and system testing here. All of the different kinds of specialty engineering testing—whether it is security, safety, reliability, capacity testing, load testing, performance testing, you name it—all these can happen in an incremental manner. Even though a lot of the especially engineering, the quality attribute-level testing really does occur at the system level, and you can't finalize the testing until the entire system's built, you can start.

Suzanne: But you still learn a lot about the system.

Don: Yes. You can start. You learn a lot about the system, and you can definitely start a lot of that testing earlier on. Once again, the earlier you find the defects, the more time you have to fix them.

Suzanne: One of the aspects of the shift left, in any of these models, is that you are testing increments earlier, and so there is not as much code. It can be easier to analyze when you find a defect. *Where is it? Where did it come in*? It's a shorter timeframe. Many of the activities that actually take a lot of time when you do all your testing at the end don't take as much time per defect. So, that has got to be a benefit.

Don: It is definitely a benefit, perhaps less so to the traditional tester who is finding the defects, but certainly to the developer who has to understand what the defect is, localize where it's at, figure out how to properly fix it, and then fix it so that you can then rerun some regression tests.

Suzanne: And so you don't break anything else. That is always one of the important aspects.

There is one more model, which happens to be…

Don: The model-based, shift-left testing.

Suzanne: Sorry. I couldn't resist.

Don: This is, in fact, sort of cutting edge when it comes to shift-left testing. As we mentioned before, the advantage is that we can start finding defects in our models. Now we have, in the past, found defects in the models by things like reviews and inspections and static analysis of the models and so on, but there is really no reason why we can't also use testing if the model is executable.

The model really becomes, if you will, code at the next higher level of abstraction. Once you have that kind of a model, it makes no sense not to start testing it. The real issue, however, is that most of the tools that currently exist supporting testing, they are all organized around code. On the other hand, you don't necessarily need tools to do this kind of testing.

You can actually, if you will, execute the model using the wetware computer between your ears. You can go through…

Suzanne: Oh, that old thing.

Don: Yes. That old thing.

You can use sequence diagrams or activity diagrams or state charts, or things like that. By executing those in your head, by walking through the models if you will, you can actually see that each one of those sequence diagrams is a test case.

Suzanne: OK. So, you can do different things to actually execute in your head, and then translate that into executing with the model itself if the tools are available to do that.

Don: Yes, and there are other kinds of tools you can use. One of our customers, for example, takes concept of operations and uses nothing more than Flash animation to animate those concepts. Then each one of those animations, the storyboards as they call them, becomes a test case that can be demonstrated in front of the subject matter expert who can then access the oracle of the test telling you whether or not the model is correct.

Suzanne: That is a very cool way of doing it. I never thought of that one. All right, so what is the current state of the practice in this arena? We have talked about the executable model testing being on the cutting edge. We have talked a little bit about this sort of traditional being almost a complete transition, but it sounds like there are still some barriers that we need to think about in this area.

Don: I think it largely is the case of whether or not you are talking about an Agile or DevOp-type application or not on whether or not you make it to that third level. Practically anyone who is building a large, complex system is at the very least, developing that incrementally with fairly large increments. If you are doing that kind of incremental development, then you are getting the benefits of that second level of shift-left testing.

The real issue, I think in my mind, is that while people are starting to make this shift, definitely in the Agile community, not everyone quite on the testing side of it realizes what is going on here and the benefit. Then, when it comes to DevOps, that is very popular when we are talking about software applications running in data centers…

Suzanne: IT applications.

Don: IT applications, MIS [management information system], where you don't have to worry about hardware. Where we don't see the DevOps testing version, and for that matter, even to a certain extent, the Agile testing as much, is when we are talking about embedded systems; when we are talking about things like weapon systems or aircraft or cars and things like that...

Suzanne: Agile, in general, is having…That is a more challenging environment for…

Don: Exactly.

Suzanne: I am working with a customer in the DoD space that we are very excited about because we are doing scaled Agile kinds of activities with them. Their testers are in every class that we are offering. There is a least one tester in every class. They are very anxious to get started in this new way. I found that to be very positive that this very complex, embedded system, the testers are there with the developers trying to make this happen.

Don: What I tend to see is that the software testers tend to be very interested in it and work to incrementally develop and test the software. What I have yet to see is very much in the way of systems testers in the Agile community adopting shift left. For that matter, the same sort of thing applies when we are talking about operational testers in the DevOps community.

Suzanne: Operational testing is tough, especially in the DOD environment. There's a lot of disablers, I'll call it, to them participating in some of this the way they might want to actually. The DevOps, I can't remember exactly where I read it, but I know something came across my feed, the latest thing is Dev Test Ops. So, they may be hearing you.

Don: There are a fair number of people out there who are talking about testing in the DevOps world, but once again the emphasis is very much on the MIS applications so far. One of the active areas of research, I am working with some people here in the SEI, is looking at testing in the DevOps world in the embedded application space, in the systems testing world. Something that is just starting, once again, but no one really knows how to do it well yet.

Suzanne: We will look forward to results of that research.

These four approaches, there is some obvious connections. If you are doing a big waterfall kind of development, then you are going to need to apply the first approach because that is the one that… But are there some other heuristics about what kinds organizations respond to the other approaches to shift-left testing?

Don: Well, I would say that every organization benefits from the shift down to the unit and integration testing and the API-level testing instead of waiting and doing all of their testing at the user interface level. That is one that seems to be good for everybody, regardless of what you are doing.

Now when we are talking about a large system being developed for a customer where you are pretty much required to provide quite large increments at a time, you may be doing Agile testing inside the development or organization, but then it becomes harder to do the Agile, the third level of testing, when it comes all the way crossed to involving the customer.

In that case what you will often see is, in the software side of the fence, you will have Agile testing, possibly DevOps testing, but you will have the incremental testing when it comes to operational testing and the overall system as a whole.

Now when it comes to the fourth version, and that is the executable model testing, there is really no reason why you shouldn't start researching that now. As I mentioned before, with either doing manual testing if you will, using your own brain as the test environment...

Suzanne: As the computer.

Don: Or, using some simple kind of tool like Flash animation or, for that matter, using some of the tools now that work on modeling languages like AADL [Architecture Analysis Design Language] or SysML [Systems Modeling Language] or UML [Unified Modeling Language], you can start doing that level of testing. The one thing I would say though is that's going to require some training. That's going to require some thought. That is a mind shift that is quite large because most people, when they think testing, they think *I am testing software* or *I am testing the system*. They don't realize that if you've got something executable, you can test it, and you don't have to wait until the software exists.

There is also another interesting thing that makes this fourth type of testing less common. There are some people within the Agile community who see, as you say, the saw tooth version of the V-model happening so often that some of them do not see much reason to do much in the way of requirements modeling or architecture modeling. Therefore, they don't see a value in investing in testing the models that they don't want to develop in the first place.

I would say, if you are dealing with something that is safety critical, security critical, truly business critical, and you are dealing with something large and complex, it makes sense to do a certain amount of requirements modeling and architecture modeling up front before you get into the Agile increments. If you do that, then there's no reason not to start testing those models.

Suzanne: Some of the scaling frameworks are much more emphatic about intentional architecture. When you get to intentional architecture, then you can get into modeling.

Don: I think there's been a wide recognition amongst many people in the Agile community that that kind of modeling definitely has its place. You don't necessarily want to do more than is cost effective to do, but doing none is often not the wisest course either.

Suzanne: You talked about doing some of the shift-left testing in the embedded software community. What are some other areas that you hope to explore in the future?

Don: Well, yes, definitely in the hardware world and the system world. I'm not even sure that the term *embedded system* adequately covers my thinking there because I truly think that what we're dealing with when we are talking about automated vehicles, robotics, aircraft, just any of these large systems. Software is an extremely important part of it, but what our customers are buying, what they are acquiring are in fact systems in this case.

Therefore, looking at how testing is done at that system level is an important issue. Then the other, I think, really big thing is, right now, we have this major separation between developmental testing [DT] and operational testing [OT]. While the commercial world is integrating these two with DevOps, we are just not seeing that happen yet when we're talking about true systems in the Department of Defense. Yet, there clearly are benefits of integrating DT and OT. We just need to understand better how to do it.

Suzanne: And, how to disable the disablers.

Don: Exactly. Another related topic is simulation. One of the key areas that we deal with these large systems is modeling the simulation. For example, when you are doing operational testing, you can't necessarily afford to have dozens of aircraft and ground sites and satellites and everything involved in these systems of systems. So, people are starting to do an awful lot of simulations involved in the operational test. Once we have simulation tools, those also support testing directly because the simulation is an executable model.

Suzanne: Once is has been validated, then you have confidence that the simulation, if the simulation works, it actually is also validated. So, you have got a chain of evidence there that the simulation models provided…

Don: And the simulation then becomes an oracle, a test oracle.

Suzanne: Well, this is very good work and, as one of the people researching Agile enablers and barriers, I am looking forward to some of the results of some of this that apply to Agile testing. I want to thank you very much for joining us today.

Listeners who want to know more about your work can obviously start with the recent book on commonly occurring system and software testing pitfalls. It is essentially a how-not-to book and we need non-examples as much as we need examples. It will help you understand, recognize, and avoid making common testing mistakes.

There are also several testing related blogs on the SEI website. Finally, there are some testing groups on LinkedIn [Bug Free: Discussions in Software Testing and QA & Testing Group] where you can engage in often great discussions on important testing issues. I imagine, Don, you're one of the people that contribute to those on a regular basis.

Don: That I do.

Suzanne: We will include links to these kinds of resources in our transcript. I want to thank you again for joining us, and I want to thank our listeners for joining us today.

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.