Three Variations on the V Model for System and Software Testing

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.

My name is Suzanne Miller, and today I am pleased to introduce you to Don Firesmith. Don's work focuses on testing, requirements engineering, system architecting, and product lines. Today, we're going to talk about Don's latest research, which introduces three variations on the engineering V model for system and software testing. Welcome, Don. Nice to see you again.

Don Firesmith: Hi, Suz. It's good to be here.

Suzanne: Okay. Let's start by talking about the V model, which is a well-understood way of representing the systems- and software-engineering lifecycle. How did this evolve out of the traditional waterfall model of software development?

Don: Well, the waterfall development cycle is a sequential series of activities. They basically just go in one direction. You know, from one activity to the next.

Suzanne: Although you can repeat.

Don: You can repeat, but historically it's been viewed as a sequential thing. One of the things people found fairly rapidly was that there were relationships between the activities at the beginning of the waterfall lifecycle and corresponding activities at the end of the waterfall lifecycle. So, for example, when one is doing system-level requirements at the very beginning of the waterfall lifecycle, at the end you have system…

Suzanne: You have requirements verification.

Don: Yes. Requirements verification. So, the two go hand-in-hand. Similarly when you're breaking the system down into subsystems and sub-subsystems, on the other side of the fence, you have integration and integration testing. What people did was they took the waterfall lifecycle and basically bent it up in the middle, so that each of the activities on the left side of the V corresponded to a corresponding activity on the right side of the V.

Suzanne: Okay. So, this gives us the V model. We have this, and it's been in use for quite a number of years. Why develop variations on this? How can your new variations be helpful to software and system developers?

Don: It's primarily for the testers. The problem with the previous approach is it was based totally on activities. So, you have requirements engineering and system testing; architecture engineering and integration testing; and so on. What the testers are really interested in are the work products they can test, the executable work products.

It's not clear from the traditional V model what those work products are, especially on the left-hand side of the V. One of the problems with the V model is that it doesn't really emphasize testing on the left side of the V. Now, historically that wasn't a problem because you didn't have anything on the left side that you could actually test.

Suzanne: From an executable view.

Don: Yes. If testing is executing something—giving it known inputs when it's in a known state and then taking a look at what its outputs are and comparing the actual to the expected—then you can't really do that when you're testing textual requirements, for example, or simple…

Suzanne: Architectural diagrams.

Don: Yes. Visio-type diagrams, those kinds of things, PowerPoint and so on. That didn't used to be a problem, but it is now because we're finally starting to get executable artifacts on the left-hand side, executable requirements models, executable architectures. You can even do the design as PDL [program design language] or you can have model-based development.

Suzanne: That's a program design language.

Don: Program design language and so on. The old approach of not emphasizing testing on the left side of the V just doesn't hold anymore. The other problem is because it wasn't really looking at testing the left side of the V, testing was delayed until the right-hand side of the V.

Suzanne: Sure. We've seen lots of cases where you don't see any involvement planned for the test community until much later in the cycle, which doesn't give them a chance to understand what the requirements are. I mean there's a lot of problems associated with that even in the traditional V.

Don: Exactly. It basically forces one to only think about testing the software or the system once it exists. By moving testing farther forward, you start finding defects in the requirements and the architecture and the design much earlier and you avoid passing those defects on to the actual software.

Suzanne: So, you are trying to move the verification steps in general more to the left side of the V to the earlier parts of the lifecycle.

Don: Well, yes, but perhaps more specifically—since we've always had some kind of verification on the left side of the V—what we're really trying to do is add testing as one more verification approach that you can apply to the left side of the V.

Suzanne: Not just analysis, inspection.

Don: Not just analysis, inspection, that sort of thing.

Suzanne: Okay. So you've developed three variations on the V model. Tell us what they are and what each of them does.

Don: Okay. The first one is just a single V. The idea here is taking the activities in the standard V model and replacing them with the testable work products that the testers are interested in.

Suzanne: So, making it clear what those test products are that they want.

Don: Exactly. So, we're not necessarily showing on the left-hand side everything. For example, if you have textual requirements, that is not really testable. But, what we're thinking about here is, *What are the executable models on the left-hand side? What are the executable things (code and software and system) on the right-hand side?* By doing that, that really brings out the importance of the different steps in the V to the tester. We're showing the tester what there is to test.

Suzanne: And, you're showing the requirements developer and the architect what kinds of things they need to be ready to provide early so that there can be early verification.

Don: Exactly.

Suzanne: Okay. What about the double V?

Don: Well, historically we've only worried about testing the testable work products in the single V if you will. But, the test work products themselves can have defects in them.

Suzanne: Sure.

Don: Then it becomes a difficult issue of trying to figure out where the defect is. You have the possibility that the defect is in the work product being tested. It could be in the test bed, test environment, or any of the test tools involved there. For that matter it could be in the test itself, the test scripts and the test data. So, what we don't want is a bunch of false-positive or false-negative test results. So, it becomes important to think about verifying the tests just like you verify the system being tested. That's where the second V comes in. The first V gives us a work product to be tested. The second V, the corresponding nodes in it, gives us the tests that we run on it.

Suzanne: That is appropriate to use.

Don: That is appropriate to use for that work product. Then, the third V adds on verification of that testing…

Suzanne: Of the testing itself…

Don: Right. Now what we can't afford to do here is have an unlimited regression, so that you have a fourth V, a fifth V, and a sixth V. The testing has to stop somewhere. That's why in the triple V model I use the more general term of verifying the test results as opposed to testing. Sorry, [I meant to say] verifying the test work products as opposed to testing.

Suzanne: Testing the test work product, yes.

Don: Now it might involve some [extra] testing. Sometimes we're testing the test work products at the same time we are testing the actual work products.

Suzanne: This is especially occurring now that we have a much broader base of automated testing. When you automate test, you're using software to create the test. It makes sense that you need to actually verify that software as well as in case of software systems, the software that is under test itself.

Don: That's exactly right. Historically this has been something that the safety community has known for a long time. If you have safety-critical software or hardware and you test it, then anything you use to generate and test that safety-critical software or hardware is considered safety-critical itself.

Suzanne: Right.

Don: So, it is subject to some of the same problems that you have [with the software or system under test].

Suzanne: This is a new way of understanding the role of testing as well as getting some details for people that are in the test community to actually make it more actionable. So, it is obvious to see how this fits into software but beyond testing software, you're looking at systems and subsystems. What are sort of the boundaries of applicability that you're looking at for this?

Don: Actually it goes in two different directions. In one sense, we're really talking about something that is restricted to testing, so it's primarily of interest to the tester, but [also] any stakeholder who is interested in testing. As you pointed out earlier, the requirements people, the architects, and the designers need to know what kind of executable work products to produce so that the tester can actually test them. On the other hand, it's more expansive because there's no reason why this can't be done, not just at the system level, but the system-of-systems level and however far up. It just means you've got a larger V.

Suzanne: Okay. So you can start applying this. System-of-systems testing is one of the areas that is very challenging because you don't necessarily know the sources of all the system elements and there's a lot of reasons that's a challenge area. But, if you apply this kind of thinking to it, what you're saying is we can now look across a broad set of systems, *What are the kinds of test work products that would help us to understand what we actually have to do to verify that the system of systems is working the way we expect?*

Don: Exactly. Then the tough part is coming up with the funding and the people to actually do that. Since it's a system-of-systems level, very often there is no central organization…

Suzanne: Right. There's no central governance.

Don: Exactly.

Suzanne: So, that is one limitation of this kind of modeling, that you don't necessarily have access to all the resources that you need to be able to execute this. What are some of the other limitations of this testers V model concept?

Don: Well, there are three main limitations of this testing model. The first is it's a very simplistic model of what's going on in the real world. In that sense it's very much like the waterfall lifecycle. There are many different types of testing that are completely ignored by it. For example, it looks at unit testing, integration testing, system testing, and so on, but there are many other types of testing that don't fit into that sort of V shape very well.

Suzanne: It communicates easily but it doesn't communicate completely.

Don: Exactly. So, for example, safety testing, usability testing, security testing, all of the testings of the qualities, for example, are not shown here very well.

Another limitation is that because it is essentially the waterfall lifecycle bent in the middle, it has all the same limitations that the waterfall lifecycle has. The waterfall lifecycle, historically in the way it's visually portrayed, makes everything very sequential with no iteration. It is sort of a big bang all at once approach where you're doing the requirements of everything before you do any of the architecture of everything. Nothing basically happens concurrently in that kind of view of the world. Well, that's not the way things actually work in the real world. In the real world, since people are doing the development and people make mistakes and don't have perfect knowledge of what's going on, there are always going to be defects that enter into the system and those need to be iterated out and fixed.

Suzanne: Besides that you have changes in the world, so your learning [and] your understanding about what the requirements are evolves and can change as technology changes or as operational requirements change. So, there's this tension between completeness and relevance, right?

Don: Exactly. The other issue here is being a big-bang approach, that might work reasonably well with very small systems, especially if the requirements are stable, but, in the real world often we're dealing with much larger systems that that doesn't scale up to. So, you can't drink the ocean in one gulp. You have to attack the problem incrementally.

The other thing is many of these activities relate to each other, and so it's very difficult to do all your requirements before any architecture because architecture influences requirements. And, when you break a system into sub-systems and sub-sub-systems and so on, you're going to have lots of teams working in parallel with each other. So, we really need to realize that you're going to have to have some sort of evolutionary, iterative, incremental, parallel, concurrent-type development cycle; something more along the lines of an Agile view-of-the-world.

The last thing is if you use the same old approaches, traditional approaches for your requirements, your architecture, design—in other words, if you just have textual requirements with no models. If you just have, for example, DoDAF [Department of Defense Architecture Framework] diagrams…

Suzanne: What I call a document-centric approach. It's not executable. Everything is represented in text and diagrams that are not going to give you that dynamic view of the system.

Don: Not only that, it's very informal. So, it's missing that kind of information. These kinds of models and texts are very static. There is nothing, in fact, to execute. If we want to get the advantages of moving testing earlier, if we want to be able to use that very important verification technique, then we have to make sure that we are producing executable models on the left-hand side of the V, so that we have something to test.

Suzanne: So, something along the lines of models produced by AADL, the Architecture Analysis Design Language, that Peter Feiler and his crew have been working on. We've talked about that in some other podcasts and those other languages, SysML, those other languages that allow you to create something that's executable. Those need to be adopted for this technique to work.

Don: Those are great approaches for architecture. The interesting thing is we have exactly similar types of [requirements] modeling languages. SpecTRM-RL, for example, is a requirements language. As soon as we start having models that are well defined, and we start having some sort of state model, then we have something where a tool can step through and produce something that is executable. And, we've got something that we can work with.

Suzanne: Something you can analyze, not just inspect.

Don: Right. Even in cases where you don't have those kinds of languages; if, for example, you have a requirements prototype, it could be something as simple as a user interface with yellow stickies on the board. If you use a human as your compiler, and step through how a user would use it…

Suzanne: Yes. We've done that, yes.

Don: You can test that user interface. So definitely we're moving into…

Suzanne: So, it's really more of a mindset shift than just the mode, if you will, of thinking about this.

Don: Exactly.

Suzanne: It's moving to something where, *How do I make this executable, even if it's just Post-It Notes, and Don and Suzie moving things around on the board.*

Don: Exactly. But, it does have the advantage that it does tend to make us think less informally, more critically about what we're doing. That forces us to not leave important things out.

Suzanne: So, this work is out at the beginning of its own lifecycle. What is your future direction? How have you incorporated this into your other work?

Don: The V models are something that I've incorporated into a chapter of a forthcoming book on testing pitfalls. It's a nice way of looking at some of the things that you need to test. My main interest, in fact, has turned from these V models to the various kinds of pitfalls where people basically end up with a testing program that is less effective than it should be and lets more defects flow through.

Suzanne: And we've talked about that in another podcast.

Don: Exactly. We did a couple of blog entries and a podcast back in the June and July time-frame on these pitfalls.

Now the book is basically done. It's called Common System and Software Testing Pitfalls. It's essentially a how-not-to book instead of a how-to book. You think of these as anti-patterns when it comes to testing. Right now, we've documented 92 different patterns, 92 different pitfalls in 14 categories. And, the book covers these in terms of what the title and description of these pitfalls are, potential applicability.

Suzanne: So, you are using the pattern framework as a way of expressing all this information, so it will be very familiar to people that are already using design techniques and things like that.

Don: It uses a very standardized approach for all of them. So, since the book is basically done now, where the future lies is in maintaining this anti-pattern language, this catalog. In fact, we've identified, since the book was finalized, four more pitfalls including one new category of pitfalls.

Suzanne: There's always a new way to fail.

Don: That's true. There are many more ways to screw up than there are to get things right. And unfortunately people very commonly find four or five of them on just about every program that they are on.

Suzanne: Yes. Well, that's why they're common pitfalls instead of uncommon pitfalls.

Don: Yes.

Suzanne: All right. Well, I want to thank you for joining us today.

I think that this way of looking at testing is going to expand the way not only some of our testers look at the world, but equally important the way the requirements analyst and your architects look at the world. I think this is going to be a great contribution. Thank you for joining us today, Don. I look forward to what you have coming up.

Don: Thank you. My pleasure.

Suzanne: If you'd like more information about SEI's current research, you can download all of our technical reports and notes at http://resources.sei.cmu.edu/library/.

You can also check out Don's blog posting on this topic at blog.sei.cmu.edu. Just click on Don's name in the right-hand authors column.

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 e-mail us at info@sei.cmu.edu. Thank you for listening.