Quality Attribute Refinement and Allocation

featuring Dr. Neil Ernst as Interviewed by Suzanne Miller

Suzanne Miller: 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 Dr. Neil Ernst, a fellow researcher who works in the SEI's Architectural Practices Initiative. Neil has been a guest on our show before when he talked about a recent field study that he conducted on technical debt. In today's podcast, we would like to talk about his recent research in quality attribute refinement and allocation.

Before we begin, let me tell you a little bit about Neil. Here at the SEI he researches the intersection of requirements engineering, quality attributes, and agile and iterative development. This includes developing new theoretical frameworks, conducting empirical studies, and communicating results to the wider community. Prior to joining the SEI, Neil was a research and teaching fellow at the University of British Columbia in Vancouver. He earned his doctoral degree in software engineering from the University of Toronto. Welcome, Neil.

Neil Ernst: Thank you.

Suzanne: I do research in iterative development, agile methods. You are doing research in this area. Out in the community we talk about the quality attributes of software. Actually the community mostly talks about the *ilities*. At the SEI we have given it the name of *quality attributes*. For those in our audience who are not familiar with that term, tell us a little bit about quality attributes. Why do they matter? How do they interact with software architecture, which is the architectural practices initiative, right?

Neil: Right. These are also called nonfunctional requirements.

Suzanne: That is another term. Yes.

Neil: That might be the more common one. I am not sure. The way we see it is these are sort of qualifications on the system's functionality. If the system has to have some kind of login function, then the quality attributes specify things like how fast that should react and …

Suzanne: Often quality attributes add constraints to the system.

Neil: Yes.

Suzanne: You can't just do anything. You have to do it within this performance constraint, this security constraint, this usability constraint. These are all things that would be quality attributes.

Neil: Right, and I think the third one—I have got it written down here somewhere so I don't forget it. Yes. I guess so you said *constraints*. I think *constraints* the way that we use it is more like *the design decisions where you have no choice in the matter. The language is Java, there is no decision to be made there.*

I think the interesting thing about the quality attributes is, well, for one, I think they tend to receive less attention than some of the functional aspects. So management is more interested in, *Does it do x and y*? Whereas from an architectural perspective, the quality attributes tend to be the ones that have the widest impact on the architectural decisions you make. For example, I think you can probably choose a number of frameworks to implement a login screen, but if you are concerned about performance, then maybe that narrows down the decisions that you can make about which one you are going to use.

Suzanne: It reduces the trade space.

Neil: Right. Exactly. I think the other aspect there is—and I think this tends to sort of be forgotten—is you can't just say *I want performance* or *I want security*. These are really tradeoffs that you have to make. I think being a good architect is about understanding which of these tradeoffs you are making and which ones you should be making.

Suzanne: Some of these tradeoffs are actually antithetical attributes. The classic one is *security* and *usability. When I have to do a 14-character password with all these attributes, it is not as easy for me to remember as a user as something that could just be alphanumerics*. Somebody has got to make that tradeoff between which of those is more important in terms of how I architect it and also how I implement.

Neil: Right. Although, I was using a new bank site yesterday, and they had a 6-to-8 character restriction, and it could only be alphanumeric. I thought I was in the field. This is not a good idea. This is going to get hacked easily.

Suzanne: They have really downplayed the security aspect and maybe not hit the right balance.

Neil: Exactly. I think from an architectural perspective, it is really about—and this is sort of the SEI's architecture approach—to start with first of all understanding what business goals the system has. Surprisingly often that isn't very well understood. From there trying to understand what the important quality attributes are for the system.

Suzanne: Let's take this into the iterative development arena because you recently co-authored a technical report that examines the use of these attributes in iterative software development. From that study, what is the current state of the practice? What are the challenges to using these quality attributes in iterative software development?

Neil: I think the challenge with quality attributes is that they tend to cross-cut a number of different functional domains. You might be familiar with Conway's Law, which says *the structure of the organization will be replicated in the architecture of the system*. You end up with a database, backend, some kind of middleware, and some kind of UI. Those are assigned to three different groups.

If you have a concern about performance for example, who is responsible for that? Then the other saying is, *if everyone is responsible, then no one is responsible*. You end up with performance gets relegated to, *let's fix that at a later point*. That is really the wrong time to be thinking about that because this is a decision that will have impact on how you are going to actually architect things.

Suzanne: In your report, you talk about some methods for refining quality attributes and for talking about them. What do you mean by *refining* because that is a term that we don't always hear in relation to quality attributes. What does that entail? Then talk about some of the methods.

Neil: Maybe I will just step back briefly to say that the way that we think about quality attributes, for example if we are doing architecture analysis, is as these seven-part scenarios. A scenario is like a use case or whatever, a way of testing the design of your system. We say it is seven parts because it has different aspects to each scenario.

I think the three ones that are important for this study were the stimulus aware, *what is instantiating or causing this to happen*?

The *context*, so in what environment…is this normal operations or is this some kind of fire-fighting environment that you have….not literally firefighting, but something's gone wrong...

Suzanne: Or degraded operations

Neil: Right, degraded operations. Better term. Thank you.

Then, finally, *what is the response measure*? We are pretty keen on having some kind of numerical or quantifiable measure of *yes, this scenario has been satisfied*. I guess here I could ask you, these scenarios to me have always sounded a lot like what you see in behavior-driven development approaches or acceptance testing.

Suzanne: Acceptance-test-driven development.

Neil: Right. In the sense that if I am going to accept this system or if I'm going to say the architecture is sufficient, it has to be able to deliver performance under a second or something like this.

Suzanne: What I see in iterative development is incorporation of these kinds of quality attributes into the definition of *done*. In iterative development and agile in particular, you will have acceptance criteria for functionality. A lot of times the acceptance criteria doesn't really deal with the nonfunctional requirements of the quality attributes. The way that we recommend people get at that is that the definition of *done* includes all those things that you want everything to meet.

If you have got a performance issue or security, you want to figure out what that is. Then, of course, you have got to figure out how to test for it, how to verify it, all those things, but you don't allow it to slip under the radar. You make sure that it is explicit. That is one of the things that we are finding is useful.

We are also finding, one of the interesting things is, the nonfunctional requirements have been picked up by a couple of the sort of Scaled Agile framework developers, but the quality attributes, I get a lot of deer in the headlights. It is an SEI-centric kind of term. In practice I don't care what term you use as long as you are paying attention to it.

Neil: Right. Absolutely.

Suzanne: So the refinement aspect of quality attributes, are you using the scenarios to figure out how to tweak those quality attributes so that they meet some new stimulus, so that they are able to create some new response that may not have been part of the scenario before? Is that the general approach?

Neil: I think going back to what you were just saying, it should be made explicit, but now the question is, if you are doing iterative development, *well, when do I do this? Like if I have a security response measure that says no one should be able to break in, well, do I do it, in this sprint or the next one*? Of course, these iterations vary widely in length and size.

Suzanne: Yes, we found that too.

Neil: Part of the technical report and the blog post looked at different ways that people refine or slice user stories in practice. For example, there are a number of writers out there on blogs and in the community that talk about—and this is a common problem in functional and nonfunctional quality attributes stories—*How do I take this big thing that I was given and turn it into the appropriate chunks*?

Suzanne: Alistair Cockurn talks about lamination. *What us the next smallest increment of value I can deliver and layer on top of.* He thinks about it as a laminating process. That has been a useful metaphor, but it is still hard. Lamination is not an easy thing to be doing. If I can just cut one panel through, right, that is easier.

Neil: Part of what we did is just survey the different approaches that people had for doing slicing in functional areas. The common one that people tend to do—I think it is part of this Conway's Law idea—is sort of this horizontal slicing. Horizontal, if you think of a layer diagram, you slice based on *let's do the database part in this sprint, and let's do the*…I'm simplifying obviously. That horizontal slicing people have sort of now realized it is a very bad idea.

Suzanne: Yes, it's not recommended by most of the agile thought leaders.

Neil: Right, because you end up with these weird silos and now you are like, *now how do I get this layer to talk to that layer? That wasn't part of any of the slices.*

The other, maybe more preferable way to do this is the vertical slicing where you say, *well, let's just start with a very simple case login. Get the data from the database, or whatever*. It's not very fancy. You don't have any of the UI Chroma in there, but it's locked end-to-end. You know, *OK, this is all working together*.

Alistair Cockburn, you mentioned him, has this other—I think it was his idea—of a walking skeleton. You end up with a very simple thing that you then flesh out later and add the other elements that you are interested in.

Suzanne: The walking skeleton idea is important, and that relates to architecture as well. That is really an architectural construct. *If I haven't gotten some good ideas about how the architecture could work, then I am going to be making decisions locally that may make me not able to walk anymore* to continue that metaphor. There are a lot of architectural implications for that.

Neil: That refinement is essentially some kind of decomposition slicing of the bigger quality attribute requirement (QARs) into the smaller pieces that will be iteration size and length.

Suzanne: We are seeing that in the DOD environments that are trying to adopt agile methods. The security one is the one. There are organizations—back to Conway—there are organizational constructs that are responsible for certifying a system for operation in its intended setting. There

are tons of security requirements that go along with that. In a traditional development, those don't get tested and really dealt with until the end.

With iteration, how do I slice that security requirement into something that is either useful to implement or, hopefully, not just useful to implement, but also useful to verify at that point. *When can I say I am done touching that because I can't verify it until I'm done touching it.* There are a lot of implications to that refinement aspect. That arena is maybe more obvious than some of the others.

Neil: I think one of the dangers of this horizontal slicing approach is that, if you say *performance is important and I know these three groups aren't going to cover it, let's assign it to some kind of architecture group*, now you have his separate problem of, *well, no one respects the architecture group because they are not writing code. They are astronauts* to use Joel Kolsky's phrase. I think that can be done in such a way that it doesn't have to happen, but it's...

Suzanne: It's harder.

Neil: Yes, right.

Suzanne: It is harder because as you say, there is not as much, it's not a team that's looking across the layer, that's looking around the layers. They're looking just across their own layer, and that creates a different kind of teaming environment.

This allocation to iterations, have you found any patterns that seem to be more promising than others? We talked about the vertical versus the horizontal, but are there any other aspects of refining into iterative elements that you would recommend to current developers, students, etc.?

Neil: I think the quality attribute scenario is essentially this vertical slice essentially. Now the question is how big a slice of this cake, if you will. *Are we going to take? It should be big enough to finish or small enough to finish in one iteration so we can say, hey, we finished this*. We went out and talked to three different organizations that were dealing with these problems. They had their existing practices. They were doing iterative development of one kind or another, and we wanted to know, *well, how do you handle things like performance requirements*?

This one example was a financial services company.

Performance is very important to get the orders back and forth. What they do is what I think its Martin Fowler talks about this as threshold testing or something similar. The link is on the blog post. The idea there is you do this thing called ratcheting. Instead of starting with the absolute, essential requirement that this system has to meet when it is delivered to the customer or whoever, for example, the order has to be processed in under a half a second. You can ratchet to that level by the response measure, in this case, by starting off with something simpler like, *let's*

get this done in 10 seconds, or maybe even the simplest thing is just *let's get this done*. Then as we go, [with] each iteration we ratchet or tighten the performance threshold until we are at a half a second.

Suzanne: This is the sort of truest form of iteration where we intentionally know we are not delivering the requirement in iteration one. We are going to incrementally iterate to get up to the point of performance or whatever that we need. We are finding that that is one of the things that's a difficult shift for organizations moving from traditional to iterative is they don't think about the fact that they actually can address this again in another iteration if they are mindful about how they design it. That is the key to ratcheting or any of these strategies, *I know I am coming back to it so I design it in such a way that that is feasible*. So loosely coupled, all those other kinds of things we talk about are important if you are going to use that strategy.

Neil: Absolutely. I think the danger from an architecture point of view is if you set the initial measure or ratchet too high, you may end up choosing an architectural approach that precludes you from getting to the ultimate goal.

Suzanne: Ultimate goal. Local optimization.

Neil: Right. Right. Exactly. I think the key here is to understand the response curve. If you said in the performance case, it has to be half a second, and you start at 10 seconds, well, the assumption there is that it is indeed possible to get this with this architecture from 10 to .5, but that's not always the case, right. It's not necessarily linear. In fact, it's almost certainly not linear in the case of performance. At some point you'll run into an obstacle that prevents you from getting to the desired criterion.

That is ratcheting on the response measure. The other things that we found these organizations doing was ratcheting on the stimulus. For example, you can assume if someone is sending an order in through your online system, that the stimulus might be when the user clicks the button. That is sort of end goal, but if you wanted to make it simpler for the first few iterations, you can say, *well, I don't know how the UI is going to process that, but let's assume that we have got this request at the middleware layer and then go from there*. So the ratchet then is right. Let's expand it out.

The other one would be, back to the quality attribute scenario, the context. If the context is normal operations, maybe everything works fine. Now we can ratchet it to reduced operating capability or server's gone down or whatever and handle that scenario somehow.

Suzanne: Or increased volume and things like that.

Neil: Neil: Right. Right. Right. Exactly.

Suzanne: Cool.

Neil: This is how those companies are doing it, sort of assigning these things to different chunks of their…

Suzanne: What is very promising to me is that they and/or you could actually associate what they are doing to these different aspects of the scenarios, which really validates the utility of scenarios for expressing these kinds of attributes and helping people to understand that there are choices to be made all along that curve.

Neil: I would say that your point about trust, which is a big part of agile, is key. These organizations that were sort of self-contained, their customer was essentially themselves. You could trust that you are going to get the job done. It wasn't like you were asking a contractor to do it, and you weren't really sure if they are going to deliver for you. One of the keys there I think is knowing, like I was mentioning about the response curve, knowing exactly what that looks like. You might have to have some other group go off and prototype or do spike or something to say, *oh, yeah this is possible*.

Suzanne: Do the research needed.

Neil: Right, but if you don't have that level of trust, then it's going to be difficult to say, *OK, it is not what I wanted right now, but I know that you'll get there at some point*.

Suzanne: Yes. Trust happens over time. It is not something that builds right away. We have a whole bunch of things about that.

Neil: Right.

Suzanne: The takeaways I have got from this are that nonfunctional requirements, quality attributes, they have architectural implications in iterative development, but they also have implications to the iteration itself, and looking at strategies for refining how we see those quality attributes evolving is one of the strategies that we actually have an opportunity to take advantage of in agile, which we don't have as much opportunity in other methods. That is the takeaway for me.

The other takeaway is that you have got to be mindful about doing that. You can't just assume that that is going to happen, that that's really got to be something that you think about as you're putting your development strategy together, possibly your organizational strategy, going back to Conway's Law. Any other takeaways that people should think about when they're thinking about this work?

Neil: I think, back to that when do you do this, how do you do this? I think, again, the open question is how exactly, if you are in a large organization, how you structure your sprint cycles, when you do planning, and so on. I know there are approaches like the scaled agile framework that have an explicit architecture runway where someone should be or some team should be thinking about these issues. Like you were saying earlier, I think it's really key to make these things explicit, whether it is acceptance criterion on an individual story or it's a specific story in itself.

Suzanne: Or it's a definition of *done* for the release.

Neil: Yes, there's *you have to have some way of understanding when and how and who is going to be responsible for that*.

Suzanne: What comes next for quality attribute refinement? Where are you going from here?

Neil: Well, I think what we are currently focusing on is technical debt. I think a big part of this type of refinement is a source of technical debt if done poorly. We are exploring that in the context of technical debt management. Again, I think the theme behind all of this is this idea of making things visible. Whether it is when you took on this decision or what iteration it is supposed to be done in, it has to be visible if only for your own sanity so you know why you did something, if not for the next team that's going to be responsible.

Suzanne: Right, sustainers of your system. Yes, that is important. Well, I want to thank you very much for joining us today. You know I love talking about this stuff. Actually Ipek knows I love talking about this stuff. You are just learning how much I love it. We do look forward to talking to you again as this research progresses and your technical debt research progresses.

For viewers that want to look at the technical report that Neil has authored on this topic, please visit resources.sei.cmu.edu. Click on the browse by author box and find Neil's name. E-r-n-s-t is how the last name is spelled, and we will also include links to these resources in our transcript.

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. Thanks for listening. Thanks for watching.