A Field Study of Technical Debt

featuring 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 very pleased to introduce you to Dr. Neil Ernst, a fellow researcher who works in the SEI's Architectural Practices Initiative. Today, we are here to talk about a recent field study that he conducted on technical debt.

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

Dr. Neil Ernst: Nice to be here.

Suzanne: For our audience members who are new to the field, please explain what is technical debt, and why should it matter to software developers and software architects?

Neil: Sure. Well, let me start with the definition that we use at the SEI. It comes from Steve McConnell, who has written books like *Code Complete*, a pretty well-known industry thought leader. His definition was a short-term decision you make for expediency that, in the long term, ends up having these costs that are associated with it. So things become more difficult to do in the long term than they should have been.

Suzanne: So unintended consequences that you can't anticipate when you make that short-term decision, and then they bite you later.

Neil: Right. So this idea of a shortcut kind of thing. The term was first defined by Ward Cunningham, another prominent Agile thought leader. He was using it in reference to a project in 1992 that I think nowadays we call an *Agile project*, but back then it was sort of cutting-edge.

Suzanne: It's pre-Agile Manifesto.

Neil: The idea there was that he recognized that there were decisions they made in iteration that were not ideal. If you are doing iterative development, you could go back and fix these things. He referred to that, this shipping first time code as *taking on technical debt*. The key I think there is that you go back and fix it. Unfortunately what we have seen is this idea of going back and fixing it is often not taken. So years later down the line you end up with a…

Suzanne: You are still dealing with whatever effects that initial decision had.

Neil: Exactly.

Suzanne: We do want to take advantage of understanding technical debt so that we can avoid some of those consequences. This is where it intersects with architecture because architecture decisions are some of the ones that have the longest-term effects on our systems, right?

Neil: Exactly. Being in the architecture practices group, we had this inkling that quality attributes and architectural issues were probably a big part of technical debt, but we weren't really sure whether that was in fact the case or not.

Suzanne: So you conducted a field study. Tell us about that.

Neil: At the SEI we do projects with a number of different organizations. We reached out to three of those organizations to help us run a large-scale survey. We issued that survey to developers and architects, other software professionals at these organizations. We got back a number of responses. From there, we were able to extract some of the things that people had said related to technical debt. I will mention that the survey itself is available on our SEI blog site as well as a related research paper for those of you of a more academic bent. We also, from that survey, asked people if they are willing to be interviewed. We got some people to come and talk to us.

Suzanne: That is where you get the most interesting data, when you actually talk to the people doing stuff.

Neil: Exactly. And some really interesting—maybe not for them but for us—interesting, painful, experiences that we've had dealing with technical debt.

Suzanne: In talking to all those people and running the survey, what are some of the findings of the field study, in particular, things that surprised you?

Neil: We had three research questions that we wanted to answer with the study. The first was the technical debt metaphor has made a big impact in industry. People have been using it in blog posts, and Twitter conversations, and so on. We wanted to see what exactly that was, how people understood that and how they were using it in practice. The second question was related to some of the work we do in architecture, and that was *Are architecture sources in fact a big part of technical debt?*

Suzanne: I made that assertion earlier, but you actually went out and studied if that is really the case.

Neil: Yes, a criticism or critique that we have gotten is, *Well, you are architecture researchers, so of course people will tell you architecture is a problem*. Obviously we understand that, so we tried to make sure we triangulated and did not bias the survey in that way.

The third question we were interested in is, *OK, if technical debt is a problem, then how are people dealing with it? What tools and practices might they use to understand or manage the technical debt they have?*

Those are the hypotheses we had in the survey. Then we got some answers back. We did some analysis that consisted of a variety of statistical approaches. The main surprising thing for us, even though we are architecture researchers, was just how big a role people were saying architecture sources were for their technical debt.

I think there is maybe a perception that technical debt is like bad code or software defects. That is not how we would typically think of it. I think there is this bottom line that you would hope that your software developers can meet. In other words, we have compile errors. The compiler won't even let you release that software.

Then I think the next step up is just writing fairly good quality code. The compiler won't find it but you should still follow the best practices. We don't really consider that technical debt. People in the survey didn't really think of that as technical debt either, so that's kind of nice. But they did see a lot of architectural sources as big, important reasons they had technical debt now. I should mention that, in two of these organizations, they are predominantly or were predominantly hardware organizations and now have a huge software practice as with most organizations these days.

Suzanne: Right. So some of their software architecture is actually determined by their hardware architectures, and so you get these a priori decisions that are essentially being made for you for which you don't always know what the architectural effects are going to be. As you discover those, that is where that technical debt comes in because you realize that I may be stuck with this architecture, but it is going to cost me something to accommodate it as I evolve.

Neil: Exactly. So we see in some cases examples where someone chose a particular chip design for example that, five years later, turns out to have been a bad practice or a bad decision.

Suzanne: It created software problems…

Neil: Often, those type of organizations may not be as aware of the software problems or issues that are involved. *It tends to be a black box. Software will fix it somehow.*

Suzanne: *Software can always fix it. Infinitely malleable.*

Neil: Usually that is the case, but…

Suzanne: There is a cost.

Neil: Right, there is a cost, and that cost is difficult maybe to make visible to people making those decisions.

Suzanne: A lot of the issues in software are about its invisibility and making these costs visible.

I want to talk just a little about Agile practices, which are iterative. In the early days of Agile, there was some rhetoric around the need to let architectures emerge and that kind of a thing. There is a faction within the Agile community that thinks that the focus on technical debt is really just a sneaky way to get people to pay attention to intentional architecture. At the same time, what I am seeing in our work is that the role of intentional architecture is starting to actually get much more positive reaction in the Agile community. Did you see any of that in the results that you got either from the surveys or from the interviews?

Neil: We did. I think that, like anything, technical debt is context-dependent. If you are working in a smaller project, or a younger project perhaps, these issues are not as relevant. The projects we are talking about that had, I would think, the biggest problems were 10 to 15 years old. They had up to millions of lines of code.

Suzanne: Lots of legacy.

Neil: Right. I think one of the takeaways there is that this idea of scaling Agile or what it looks like for projects like that is still a relatively unanswered question. You and I are both trying to answer that one.

Suzanne: Yes, we are both playing in that pond. Yes, what we are seeing is the intersection of sort of Agile team practices with more Lean engineering practices as one of the directions that people are going. The other direction is more scaling up, sort of scrum of scrum of scrum of scrums is a different pattern. We are seeing a couple of different patterns. In both cases, there seems to be the larger the project gets, the more understanding there is that architecture is going to have an effect on your outcomes, and that you can't leave all the architectural decisions down to the team level. This is one of the reasons why right? Incurring these unintended consequences.

Neil: I think there is a distinction that I believe Martin Fowler has drawn too, which is there are these inadvertent decisions that at the time were the right decisions to make. Then the context changed, and two years later it isn't the right one. So there were some of those answers in the survey as well, but there was also a lot of answers where a common anti-pattern would be something like prototype becomes product. If you are trying to win a bid for something, you get your team together, you burn a lot of the midnight oil, etc., etc. Then there is a sort of this implicit promise that we'll fix it if we win, and that never happens, and then…

Suzanne: *We are never given the resources, time, or the tools to actually go from prototype to production quality.*

Neil: Exactly. I think one of the interesting things was just how disconnected the decision from the understanding of the problems was. So, in some cases, five, six years have passed, and now you are struggling, whereas, what we would hope you would do is sort of manage this actively and say, *Look, we took this on knowing that we would get the product out the door, but we have to go back and fix it at some point.*

Suzanne: And having actually an understanding of what those decisions were. I think one of the things that I see in some parts of the Agile community is the reliance on tribal knowledge is great until the tribe changes. Five, six years later, you may not have the people that knew why we made this decision and what we know the effects would be as well as some unintended effects are. That becomes difficult.

I actually know a few people that are developers in the Agile community that have gone back to actual handwritten notes in their comp notebooks. That is part of what they leave with the project if they leave the project because there are so many of these conversations that happen that are valuable, wonderful, give us better results for the users, but don't necessarily communicate what the next engineer needs to know.

Neil: I think one of the revelations for me in coming here, I have only been here three years, is just how fluid team structures are in these larger organizations where you can have a group of people who help win the bid. Then they are taken off and put somewhere else.

Suzanne: They go win another bid. That is a common pattern.

Neil: Or, you have got people coming and going from the project for whatever reason. Yes, that tribal knowledge just completely evaporates. Typically what happens is you only recognize the technical debt, or the management only recognizes it, when things come crashing to a halt. It is like, *Oh yeah, we should have put some money into this*.

Suzanne: *We have to completely re-architect.* It is not just a matter of refactoring, it is a matter of actually re-architecting the whole thing, and that costs a lot of money. Now that cost is visible. It is no longer invisible.

I think students in engineering should be interested in this, software development professionals, software architects. How can they make best use of the findings of your study? What are some of the takeaways? We have talked about a couple of them, but are there others that you want to highlight in terms of things that people should pay attention to?

Neil: Yes, I think the only other main highlight for me was the use of tools. Tools tend to be seen as a solution to all the problems. If you just buy this new tool. There is a lot out there. There are some really good tools. They're not…

Suzanne: There are some necessary tools. I mean, you can't do the level of regression testing that iterative development demands if you don't have good automated testing tools, for example.

Neil: Exactly. We asked these survey participants and some of the interview participants what they were doing with tools. By far, the majority of them were not doing anything with tools. Of the ones that do use tools, the tools that they were using were issue trackers, so not really technical-debt-specific in any sense. There are some who used a few different technical debt tools.

By technical debt tool, I mean a tool that will, for example, highlight code quality problems or that will identify some common architectural problems like modularity violations and so on.

Suzanne: What would be commonly called static analysis tools.

Neil: There are a lot of static analysis tools. There are a few that go into more of what I would call the architectural realm as well. The idea behind these tools is essentially to try to put a number to each of these problems. For example, a problem might be that your switch statement doesn't have a finally clause or whatever and that costs five minutes of developer time times whatever the developer costs. Those tools are not widely used right now, at least in the people who responded to the survey. I think that's for two reasons. One is people just don't have time to figure out how to use these tools. I think that's common to probably any tool…

Suzanne: It is an investment.

Neil: Right. The other problem is that these tools need to be contextualized. Coming back to the software quality attribute idea, your project may be a performance critical project and in which case your technical debt you're concerned about is really performance-related. Mine might be maintainability or something else. And if the tool is…

Suzanne: both of us have to worry about security.

Neil: Yes. Right. If the tool isn't contextualized for whatever you are interested in, it ends up just generating a bunch of fairly useless false positives. At that point, people just say, *Forget it, I…*

Suzanne: They don't have the time to muddle through what is meaningful out of the output. Yes, we have seen that.

Neil: I think one of the takeaways is I have used these tools myself, and I do think that they are very useful. The most use comes from installing the tool and getting somebody to help you figure out how to configure it or to analyze what it is giving you back. It typically isn't that hard, but you do have to do it. Once it is done, you can probably go ahead and run with it.

One of the common use cases for this is acquisition, not like DoD acquisition, but acquiring another company. If you are like HP [Hewlett Packard] acquiring EDS [Electronic Data Systems Corporation], you don't just acquire the people or the infrastructure, you are acquiring all their code. You really want to know, *Am I buying good stuff or bad stuff*? [It is] sort of like a home inspection, I guess, if we continue with that building metaphor.

Suzanne: We are seeing more and more of that with different mergers, acquisitions, people inheriting legacy and not necessarily understanding the debt that they have incurred along with that.

So is that what you are doing next? What do you have on the plate for continuing this work?

Neil: I think the step that I am interested in investigating some more about how tools are used and how to make these configurations…

Suzanne: …how to make home inspection tools.

Neil: Right. Make them more automatable so you can download the tool from the vendor and then have it set itself up a little more intelligently than just right out of the box.

Suzanne: Give it some parameters, and let it go ahead and do what it needs to.

Neil: The simplest thing is simply, *Do you care about X, Y, or Z?* I almost said *zed* there. I am Canadian.

Suzanne: I would have interpreted for our viewers on that one.

Neil: With the issue trackers, some of my colleagues are doing work on, *Should you just throw anything into the issue tracker or should you add some special fields?* For example, the obvious field to add would be *This is technical debt. Come back and fix it later*. So we are looking at some existing projects to figure out, *Well, What are they doing and how*, so we know…

Suzanne: How do they communicate about this?

Neil: We have talked to them in person, and we know what they say the technical debt is. Now we can look at their issue repository and say, *Well, is that actually what we see?*

Suzanne: Does it reflect what they are actually thinking?

Neil: And why or why not? Finally, we are working on a few courses and other training materials to help educate people in this area.

Suzanne: This is an area that I think is one of the gaps in education in terms of undergrad, especially *get the code done*—I am so happy with that—but understanding as a professional your responsibility to not incur technical debt that is unintentional. That is one of the things that course ware is going to be useful for.

Well, I want to thank you very much for joining us today, Neil. We look forward to talking to you again as your research progresses and we have other things to talk about in terms of your studies.

If you want to view an SEI blog post that Neil authored on this topic, please visit insights.sei.cmu.edu click on the author box and find Neil's name under E for Ernst, E-R-N-S-T. 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 contact us at info@sei.cmu.edu. Thank you for listening, and thank you for watching.