Security Pattern Assurance through Roundtrip Engineering

featuring Rick Kazman 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 located on CMU's campus 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. I am a principal researcher here at the SEI. Today, I am very pleased to introduce you to Rick Kazman, a senior researcher here at the SEI who is also a faculty member at the University of Hawaii. And yes, he does get to go back and live there part of the year.

Rick's research focuses on software architecture, software engineering economics, design-and-analysis tools, and software visualization. He has authored numerous technical reports and is coauthor of several books including *Software Architecture in Practice* and *Evaluating Software Architectures: Methods and Case Studies*. He's a very busy man. In today's podcast, Rick and I are going to be talking about new research that he is conducting on the use of architectural patterns, the challenges associated with them, and how they can be used to achieve system qualities through roundtrip engineering, a new term in our lexicon. Welcome, Rick.

Rick Kazman: Thanks for having me, Suzanne.

Suzanne: Absolutely. Let's start with patterns: architecture patterns and how they're used to design software systems. What's the overview of that approach?

Rick: Probably most of the listeners are familiar with design patterns since the 80s the "Gang of Four" book has popularized that notion. But, patterns in general are a useful concept for design at any level of abstraction. What we found over the years in teaching architecture design and architecture analysis is that almost nobody starts architecture design from a blank page. Almost nobody does it from scratch. People use patterns, well-known, well-understood, patterns to achieve their goals. Sometimes they do this subconsciously. Sometimes they do it based on their gut, their experience. Sometimes they do it consciously, looking up repositories of patterns.

There are lots of these published on the web. Microsoft has published a repository of patterns. We've published patterns. They are out there. We teach it, and it's a really useful tool for designers.

Suzanne: You are trying to overcome some challenges that come along with dealing with these patterns. What are some of the challenges that you're trying to overcome?

Rick: Well, there are always challenges of understanding and interpretation. You get 10 people implementing let's say the layers pattern, which is the most common architectural pattern of all. They will implement it in 10 different ways. Some people will allow upcalls. Some people will allow downcalls. Some people will allow horizontal calls, layer bridging, side cars; all kinds of variants of what you would find in the textbook description of the patterns. There are differences in how they're understood and instantiated.

Then, maybe a more insidious problem is that once you, the architect, have designed a pattern and you hand that over to the developers to be realized, they are free to do whatever they like. That may or may not follow your design intent.

Suzanne: That can lead to problems with quality attributes, with system qualities.

Rick: Absolutely.

Suzanne: Because you have designed into that pattern some things that you want to happen with the system. If they don't follow the guidelines—essentially that go along with that pattern— then you could break the assumptions about those qualities. So, that can be a problem at the system level.

Rick: That will undermine the integrity of your system. Unless you, the architect, are constantly checking, A) you don't know what they've implemented, and B) chances are the intent will at some point be undermined maybe in the initial development, maybe over the course of years of maintenance.

To give an analogy, if I was a structural engineer, a civil engineer, designing a building, I would figure out, based on the anticipated weight of the building, how strong the foundation needed to be and how deep, how thick it needed to be. When the builders poured concrete for that foundation, I wouldn't simply trust that it was strong enough. I would take core samples, and I would have those samples analyzed. Unless they were strong enough, I wouldn't proceed. We don't do that in software engineering, at least not on a regular basis.

Suzanne: To continue the analogy a little farther, if somewhere in that early process someone decides that instead of 10 stories this needs to be a 15-story building, then you are going to have to look at that foundation again because you didn't design the foundation to meet that set of attributes.

Rick: Absolutely. When we evolve software systems, we do that all the time. We throw in extra stories without ever reconsidering whether the infrastructure is adequate.

Suzanne: So, you've got a new concept for helping to deal with this called round-trip engineering. So, tell our listeners about round-trip engineering and how this addresses the challenge that you're talking about.

Rick: Sure. So, there are two kinds of engineering that we frequently do when doing software maintenance. One is forward engineering. That's adding stuff. So, to continue with the example, if I had a layered architecture, I might be adding some functionality in layer one, layer three, or layer four. I may be adding a whole new layer onto the top. I may be modifying some relationships between the layers. That's forward engineering: the stuff we do, the bread and butter of software development. You add stuff. You add features. You add functionality.

Similarly, when we want to make a change, if we're perhaps new on a project or we're no longer sure exactly what the structure of the code base is, we'll often do reverse engineering.

Suzanne: Or, adopting a legacy system.

Rick: Absolutely. We'll often do reverse engineering. That's where we look at the artifacts and try and make sense of them, maybe [by] building some pictures, some structured diagrams of what's there. You've got forward engineering, reverse engineering. They are relating your understanding to some architectural artifacts in two different directions.

If you can combine those into a single toolset and a single methodology, what you've got is round-trip engineering, where I can be developing stuff, and that will hopefully be reflected in the architectural representation. I can check at any point whether that architectural representation matches the stuff that has been developed, the stuff that I want to develop. That gives you control over the whole round-trip, and that's what gives you predictability.

Suzanne: And confidence.

Rick: Confidence. Yes. That's the reason that the structural engineer does a core sample.

Rick: These are our core samples.

Suzanne: How far has this concept gotten out into the world so far? You blogged about it. What other kinds of things have you been doing with it?

Rick: This idea has been around for a long time. So Rational Rose—10, 15 years ago at least—had at least primitive notions of this capability. Obviously, it could help you with forward engineering. It could help you figure out what was there. But, overall, the adoption of this idea has not been strong, I think, for a couple reasons. One is that the reverse engineering tooling has and continues to be fairly primitive and awkward and not very useable. The other is that we as an industry continue to focus on adding stuff: adding functions, adding features with really little regard for tomorrow. It’s a kind of shortsightedness that we’ve gotten away with, kind of, most of the time, sort of.

Suzanne: Until the building wants to fall down, until later. That’s actually part of this whole thing is that I think we have a better understanding than we did even 10 years ago of just how long our business systems, our military systems, any of our critical infrastructure systems are actually out in the field. They’re not going to be replaced. They’re going to be added to. They’re going to be evolved. To lose sight of the patterns that were used and how they were implemented, it really does lead us to problems with how the quality attributes we thought were there are actually going to either remain there or get deleted or somehow impacted.

Rick: Yes, and a lot of times that important knowledge about the system, about the structure of the system, is in the heads of very few people.

One of my favorite personal anecdotes about that is the very first time I did an ATAM, an architecture tradeoff analysis method, with a real company. The first day that I went in there, the lead architect wasn’t there. Every time I asked somebody a tough question, the answer was, *Ask Cal. Cal knows the answer to that*. I immediately thought, *What happens if Cal gets hit by a bus?*

Suzanne: We like them to be hit by a lottery truck. That’s a much more positive spin on it.

Rick: Yes. In the absence of some kind of documented representation or the ability to get it via reverse engineering, you’ve lost a huge resource if you’ve lost Cal.

Suzanne: It’s the tacit knowledge versus the explicit knowledge trade-off.

Rick: Yes.

Suzanne: You spend some time and money documenting and helping people to understand and make explicit the knowledge and the understanding. What you lose in time, you gain in portability. You gain in forward understanding and efficiency later on.

Rick: Future Efficiency. Yes.

Suzanne: You also gain an understanding of *No, we shouldn’t touch this*.

Rick: Right.

Suzanne: That’s one of the big things I see in these kinds of architecture documentation sorts of things.

Rick: Overall, what you want to gain is confidence in whatever decisions you have made. You might make the decision to hack away and not re-inforce the infrastructure. That's a conscious decision, and that's OK. Sometimes you say, *We have to meet this deadline in six weeks, and we'll go clean up later.*

Suzanne: Yes. As you say, it's an explicit decision. It's made with knowledge of, *you're going to have to pay that back later*. Another concept that we use here is technical debt.

Rick: Absolutely.

Suzanne: It's that idea of knowing that you're going to have to clean things up later that you implemented in a way that is not going to really hold on for a long time.

Rick: Yes.

Suzanne: So this work, I know you don't work alone. [For] most of this work you have very interesting collaborators. Tell us a little bit about the people you're collaborating with on this project.

Rick: Sure. So I've collaborated with a couple people outside of CMU on this project. One is Dr. Amnon Eden from University of Essex in England. He and his research group have developed something called a two-tier programming toolkit. The two tiers are the design tier and the implementation tier. What we were just talking about.

The toolkit helps to automate roundtrip engineering. There's a forward engineering tool where you can create architectural patterns. You can associate those patterns with code. So essentially, you can say, *I'm doing single access point pattern. Here are the major structural components. Here are the relationships, and this bit of code implements this part of the pattern. That bit of code implements that part of the pattern.* So, you associate it explicitly with your code base.

The tool helps you keep those aligned. It will tell you when the implementation diverges from the specification. You can either accept that, or you can fix the implementation. Or you can say, *No, in fact, I don't want to implement the vanilla single access point pattern. I want to implement my own little variant of it.* Again, you're doing it with knowledge, and it's documented.

It also supports reverse engineering, so [it's] helping you to locate those patterns in the code and identify them. So that's Dr. Eden and his group. That's one group.

There's also Dr. Jungwoo Ryoo at Penn State. He's been helping in one specific use of the toolkit where we've been trying to catalog security patterns. And try and understand system security or their security design from the architectural perspective. And find places where those patterns have been improperly implemented and not implemented—something like that.

Suzanne: Places where the patterns inherently conflict can also be some fun when you get into security.

Rick: Always good.

Suzanne: Where do you go next with this? I think the tooling is going to be an important element of this, similar to how automated testing becoming easier has made regression testing and some of the things that we do in the agile implementations more feasible. It sounds like this is another area where the tooling improving is going to make a big difference in the implementability of these concepts. Is that the main focus?

Rick: Yes, so the tools are research prototypes. We're working on improving them, hardening them, expanding their capabilities. Also, we've been working on improving the reverse engineering side of the tooling. Right now the reverse engineering side is largely human driven. The human says, *I think pattern X is implemented by these files or those classes.* Then, the tool helps you to determine.

Suzanne: To catalog that.

Rick: To catalog it and determine whether that is really the case. What we're looking at is some machine learning tools that help you find the patterns, so more heuristic matching. Fuzzy matching, if you will.

Suzanne: For a lot of coders, it's going to have to be fuzzy.

Rick: There's a lot of fuzziness out there. The idea is to help guide the analyst and give them a leg up because there's a lot of code in modern systems. And, you don't know what you don't know. The tools will help you find out what you don't know.

Suzanne: Well, that sounds like it's going to keep you busy for a little while working on this. Along with all the other projects that you're involved in. I want to thank you very much for joining us today, Rick. It's always fun to talk about what you're doing because it always makes me think of new things.

Rick: Oh, my pleasure.

Suzanne: For more information on the round-trip engineering approach, please check out Rick's recent blog post at blog.sei.cmu.edu. Then click on Rick Kazman's name—that's with a Z— in the author category.

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.