
# Software Architecture Patterns for Robustness

*featuring Rick Kazman as Interviewed by 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. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.*

**Suzanne Miller:** Welcome to the SEI Podcast Series. My name is Suzanne Miller, and I am a principal investigator in the SEI Software Solutions Division. Today, I am joined by my friend and colleague, Rick Kazman, a visiting scientist in the SEI Software Solutions Division. We are here to discuss software architecture patterns in support of developing and applying robust, complex systems. These patterns were recently explored in the fourth edition of *Software Architecture In Practice*, which Rick co-authored with other friends and colleagues of ours, Len Bass and Paul Clements.

Welcome, Rick. It's good to talk with you.

**Rick Kazman:** Good to talk to you Suzanne.

**Suzanne:** Let's start by having you tell our audience that may not have heard from you before a little bit about yourself, what brought you to the SEI, and the work that you do here. In particular, you have this unique connection to academia that not all of our staff do. Explain a little bit about your, it's not even bicoastal, I don't know what to call your situation, but we will just talk about it.

**Rick:** World's longest commute. I have two positions. I am a professor at the University of Hawaii, and a long-time visiting scientist at the Software Engineering Institute, also in the Software Solutions Division. I have had various affiliations with the SEI dating back to the '90s. In I guess 1998, Len Bass, Paul Clements, and I published the first edition of Software, Architecture, and Practice. It's now in its fourth edition, as you mentioned, which is kind of crazy to think that, I guess, that means I am old.

It has been a really interesting avenue for us for doing research and really practically grounded research. So, all along, we have done a lot of collaboration with SEI customers with real-world projects to inform and enrich what it is that we write about, what it is that we profess. So for example, the technical report on which today's podcast is based, that came out this year. But it was, in turn, based on an earlier technical report that came out in about 2009 that was based on some work that I did with an architect at Boeing on availability. So yes, we have a long legacy of doing work that has, I would say, two feet: one in the world of academia and the world of research and another firmly grounded in the world of practice.one in the world of academia and the world of research and another firmly grounded in the world of practice.

**Suzanne:** That is one of the things that I know about you is that passion for making sure that academic work has a grounding, and that we understand how to apply it, which is very much in tune with our mission here at the SEI. The architecture topic in general with the SEI is very long-lived. It was one of the early initiatives at the SEI. It is notable that we are still learning new things about architecture both in terms of where we can apply older thinking to newer situations, both software and hardware, and new thinking about, as we get microservices as a pattern, for example, what effect does that type of architectural pattern have on things like availability, robustness, deployability, security? So a focus on quality attributes, I think, is the unique aspect of the SEI's architecture work, but it is the gift that keeps on giving because there are always things for us to think about in those areas. Let's focus today on availability, robustness, and what have you been learning recently about those topics in terms of architecture patterns and especially cautions for some of our viewers who are architects that may not be aware of some of the things we have learned about. *Hey, if you engage in this pattern, it's going to reduce availability*. Or *this pattern, in contrast, may improve it*. So, please.

**Rick:** Yes, I mean a lot of people will fall into the trap of thinking, for example, *Well I am deploying to the cloud, so I don't need to worry about availability. The cloud takes care of all of that for me*. While it is true, the cloud does an awful lot for you, it makes life a lot easier than what architects had to deal with 20 years ago, there are still a ton of decisions that you as an architect need to make. Old architects, curmudgeons will talk about how there is nothing new, that it has all been done before, these problems have all been seen before. But the technologies do change, and how we think about them does change. We documented a bunch of patterns. Let's back up a second, we documented a set of mechanisms, what we call *mechanisms* in the technical report; and mechanisms is our umbrella term for tactics and patterns. Tactics are the basic building blocks of architecture, and patterns, we like to say, are made up of tactics. So we say that by analogy, tactics are the atoms and patterns are the molecules.

**Suzanne:** Right.

**Rick:** So the basic tactics really haven't changed much since 2009, since that TR [technical report] in 2009, but in working on this current report, which I should say was supported by and motivated by a government customer of ours, we realized that we did need to make that connection between the tactics, things like tactics would cover the basic ideas of, *How are you going to detect a fault? How are you going to recover from a fault? How are you going to prevent a fault?* Those are the categories of tactics. When you translate that into patterns, well, there are a lot of details. The details matter. So that's a lot of what we focused on in this TR is listing those tactics, and how to think about them, how to reason about them, how to analyze for them, but then really drilling down into the patterns.

**Suzanne:** And I am guessing that some of this, you mentioned this at the beginning, the cloud is a different context of deployment than 2009. I mean, there were cloudish things in 2009, but the ubiquity of the cloud was not present at that time. So our architectural focus was not on, *Hey, make sure that all of our architectural patterns are robust to the cloud*. Right?

**Rick:** Right.

**Suzanne:** So is that really one of the things that is driving some of the revisiting of these concepts?

**Rick:** Yes, and even things like, so let me start off with a concrete example. Probably the most famous, most common availability pattern is triple modular redundancy. That is where you have three identical components. This would be an instance of the active-redundancy tactic, and they are processing the same inputs in parallel. The idea is that if one of them goes down, well, you have still got two others. This also brings up the opportunity, and this is a part of the pattern that you often see, for a voter. So if there is a chance that one of the components could fail live, that is to say it fails but it's still spitting out something, you can look at the results and you say *Well, two of them agree, one disagrees. We are going to mark that disagreeing one as bad, and maybe do something to replace it or repair it. We are going to still continue on with confidence; knowing that these other two agree and that we can continue processing without any interruption in service*.

That is the most simple pattern, and that is the sort of thing that you can very easily do in the cloud. But something that is a little bit, let's say more sophisticated than that, would be the circuit-breaker pattern. The idea of a circuit-breaker pattern is that there are often cases where a service fails, and it takes a while to figure out that that service has failed. So some client sends a request to the service, and then they wait, and they wait, and it times out. Then maybe they send another request, and they wait, and they wait, and it times out. This might happen several times before they finally decide, *OK I need to go to the backup service or do something else. Report a failure*. In the circuit-breaker pattern, what you do is you have a monitor that is monitoring that

service, and the monitor acts as an intermediary between the client and the service. If the monitor determines that the service is no longer active, the client, which is now talking first to the intermediary, will immediately be told, *Sorry, no service today.* So it is a circuit breaker. You fail instantly, and that allows you to recover or reroute, or you know, do something…

**Suzanne:** Or make a decision.

**Rick:** Make a decision to inform the user, whatever, do something much more quickly. That is the sort of thing that a cloud doesn't just give you by default.

**Suzanne:** Right. You have to do something.

**Rick:** You have to do something. You have to think about what is the nature of the possible failures, and what would I like to do? How would I like to respond?

**Suzanne:** I am just thinking, I actually am seeing that pattern in my Internet service. Now, when power goes down in my house, I get a text message on my cell phone from my Internet provider saying, *Hey, we've detected a power outage that may be affecting your Internet service.* Then when the power goes back on, they send me a message that says, *We think the issue has been resolved in your area*. They are leaving it to me to make a decision, but they are giving me that circuit breaker, and honestly, it's gotten to where it is easier to see that than to call the electric company and say, *Hey, is there an outage in my area?*

**Rick:** Interesting**.** And you could take some recovery actions. Maybe you have got a backup power generator and you could go downstairs and fire it up, or you could just wait, or something else. Maybe you test a power wall, and you fail over to that seamlessly. That is the idea is the patterns, in general, are technology agnostic. How you instantiate them, how you realize them is going to make a big difference to things that you should and likely do care about, like mean time to failure, mean time to recovery.

**Suzanne:** Or real-time processing that can't be interrupted. Many of our government customers, whether it's the IRS not wanting the service to go down on April 15th when everybody is hitting the servers to file their tax returns or some of our military applications where there is a can't-fail kind of mission. That is one of the things that is very important about robustness and availability is that we have context throughout our world. I mean, my situation of, *Oh my, the Internet went down*, well that may interrupt us finishing this podcast, but nothing terrible is going to happen. But there are situations where bad things happen if service is interrupted, and so architects that are working in those areas really need to be aware of these kinds of patterns to make sure that they don't inadvertently cause a live failure or a system to go down when we can't afford it to go down.

Are there new things, new patterns that we didn't really know about in 2009 that have either because of the cloud or just because of other parallel processing or other kinds of advances in technology, that architects that have historically been working in these systems may not know about? Are there some new things that we should talk about?

**Rick:** Again, there is nothing that is totally new. You can always find some precedent for one of these patterns, you know, in a system that somebody built for Telecom in 1978 or something like this. There are certainly patterns that have come much more to the fore.

**Suzanne:** OK, more prevalent or need to be more prevalent.

**Rick:** Yes, with the rise of the cloud. One thing that is really popular is the health-monitoring pattern. A health-monitoring pattern, just if you think of the analogy of a health monitor, patient monitor in a hospital that is beeping and is keeping track of a patient's heart rate and blood pressure and other vital signs. This is doing the same thing for a software component, some sort of process or some runtime component that you care about, a service. It may be very simple things like the health monitor just pings the component periodically to see if it responds, or it may be something much more sophisticated looking at round-trip latency, looking at CPU utilization, memory utilization, or application-specific measures like percentage of this kind of request that it responds to, or queue lengths, things like that. This is becoming much more popular in distributed systems where you have got components that may be run by, operated by different service providers, maybe operating in different clouds, and you don't have a single way to assess the health of the system.

**Suzanne:** Gotcha.

**Rick:** So you use these health monitors to figure out, *Is the problem over here? Is the problem over there? Should I send the physicians running? Do we need to do something?* This allows systems to be much more adaptive. So again, you could probably find examples of health monitoring in the past, but it has become very prevalent. There are entire companies whose existence is in observability frameworks, which are basically the foundation for health monitoring.

**Suzanne:** I would guess that health monitoring has also become more prevalent because of DevSecOps because I know some of the systems I have worked with, that is, how you are monitoring is, *Are things getting through? Do we have too many defects going through? Are there build problems? Is this build service giving us what we need or is the latency too long in terms of how long it takes?*

**Rick:** *Are we under attack right now?*

**Suzanne:** Yes. Yes.

**Rick:** Can we  distinguish *very popular* from *under attack*?

**Suzanne:** Yes, so there are a lot of different contexts where health monitoring… I hadn't really thought about the fact that with cloud, we do have distributed providers, and so if one provider is being attacked, in some cloud environments, we have the option to switch, we just have to know. Other times, maybe not so much. But that would certainly be one that we want to have people pay attention to.

**Rick:** If your doctors didn't have all those diagnostics, they would be, you know…

**Suzanne:** They would have a hard time.

**Rick:** Yes, very little use in actually making you better. Another thing that is related to that but a little bit different, but it again has become much more foundational, I would say, in cloud environments, in service-based systems, in microservice-based architectures is the throttling pattern.

**Suzanne:** Is the which pattern?

**Rick:** Throttling.

**Suzanne:** Throttling. Oh, I thought you said thralling. OK. Throttling pattern, yes. I have heard of that one.

**Rick:** Yes, so the idea there is that you might have, let's say a service-level agreement that you have pledged to maintain, but you can only maintain that if you are not dealing with more than X requests per second. So if requests per second go above that, you need to throttle those. I mean there are other options. As an architect, I could try and increase resources, I could try and increase parallelism, increase the size of server pools, and things like that. But at some point, you may not be able to increase resources infinitely, or you may have a bottleneck that you just can't get around, and so one option, not the only option, but one option for an architect is to throttle demand. This could be rejecting certain requests: *We are not going to deal with the low-priority requests, we are only going to deal with the high-priority requests.* Or it could be, well, *These requests have already gotten so much resources, so we are now going to serve the ones that have gotten less resources*, so the throttling policy is up to you. But the core of the pattern is monitor demand in some form. Again, that could be queue lengths or average response time or something like that. Then, when that exceeds some threshold, you start rejecting, or delaying, or deferring certain requests.

**Suzanne:** Another very popular pattern in telecoms.

**Rick:** Yes.

**Suzanne:** This is when we guaranteed this much data at this speed, and if you go above that, then you're going to get throttled. They don't say it that way, but you're going to get throttled down to a lower speed until your next period of service or in case we have extra resources, we'll bump you back up.

**Rick:** Yeah, and you kind of see that in video-conferencing systems, where sometimes, rather than losing a connection entirely, you'll go to a lower fidelity.

**Suzanne:** Right.

**Rick:** And the images start to get jerky, and the voice starts to get choppy, but you can still at least some of the time make out what's being said.

**Suzanne:** Or even where you decide to turn off your video because that reduces the bandwidth. So you can get a better audio signal. We have all run into that from time to time.

**Rick:** So those are human instances, or human-controlled instances of throttling, and yes, we have all done that, especially in the last couple years.

**Suzanne:** Yes, exactly. So we have the technical reports. We have got the fourth edition of *Software Architecture in Practice*. We have courses based on that [here and here]. So those are a few of the resources that we will mention in our transcript for this. But are there other ways for people that are not as familiar with especially using these kinds of patterns in the cloud that you would recommend for architects, either architect students or architects in practice, to go find out more about how these patterns affect robustness, and things that they can do to increase the robustness of their particular applications?

**Rick:** Yes. There are a ton of resources these days on patterns available everywhere. Wikipedia, and Stack Overflow, and Reddit, you name your favorite technical community, and search on *availability pattern*, or *robustness pattern*, or something like that, you are going to find something. But I will say that we put a lot of thought and a lot of effort into this technical report. It's probably the longest technical report I've ever written, and so that is a pretty good place to start. It's a pretty comprehensive guide. We talk about what robustness is. We talk about how to specify it in terms of robustness requirements. How to create scenarios, robustness scenarios. We talk about the mechanisms, the tactics, and the patterns. We provide a whole bunch of analysis techniques, so ranging from fairly informal questionnaires and checklist-type approaches to much more formal Petri nets and Markov models and things like that. Which of course, there are entire books written on this topic, but we give an overview and then lots and lots of pointers to the relevant literature. Then, at the end, what we provide is a playbook, and a playbook is

something that we have done in the last few quality-attribute reports. The idea of a playbook is well, just what it sounds like. It gives you a set of steps that you, as an analyst, can walk through to figure out, *Have I thought deeply enough about robustness? Have I asked the right questions? Have I collected enough information to be able to do analysis? Can I figure out if the requirement that I have is likely to be satisfied by the architectural decisions that have been made so far?* So that is the very end of the report. I think that if you haven't had a lot of experience in designing for and analyzing for robustness, looking at that playbook is a nice way to structure your thinking, to organize your thoughts and your tasks.

**Suzanne:** Gotcha. So this is probably not something that is part of the robustness discussion, but it strikes me as we are talking about these kinds of patterns that for those that are in the cybersecurity community, availability is a security issue as well, and many times, our quality attributes are synergistic. Sometimes they are contradictory, but this is a case where they tend to be synergistic. Are there any of the patterns that contribute to robustness that come to your mind immediately and, *Oh, by the way, this is a really good pattern to be aware of if you're trying to build in security*, which we hope everyone is trying to do.

**Rick:** Well, for sure, as you and I already said, all of these monitoring patterns are really useful for monitoring liveness availability, but you could monitor other stuff as well. So you can monitor patterns of connection requests to determine whether those are likely attacks. Another pattern that we didn't talk about is forward error recovery. This comes from the telecom domain. That comes from the idea that you add on to the payload that you are delivering, some additional information that you can use to check the validity of that information when it arrives. This is one of the key ideas behind blockchain-based architectures. That is another pattern that we talk about in the technical report that I think you could argue that has applicability to security as well.

**Suzanne:** I am just thinking about some of the people that I work with and multiple attributes like this. If you can get patterns that support multiple quality attributes, that is a real big win, especially if they don't have to be contradictory.

**Rick:** Another one is, there is a pattern called recovery blocks, which is basically a kind of N-version programming pattern where you have different versions of the same functionality. That is something that is not often done, I think, for security. But it kind of makes sense that an attack that is successful on one implementation may not be successful on another implementation. I haven't thought deeply about this, but now that you bring it up…

**Suzanne:** Your next technical note!

**Rick:** That might be a reasonable security approach. I mean it is expensive to do inversion anything. But if you are really worried about security, and if this is really the knot of your

system, the heart and soul of your system, you might want to think about protecting it in a, let's say a less traditional way.

**Suzanne:** Speaking of your next technical note, what is interesting to Rick Kazman right now, and what kinds of things are you thinking about, writing about, getting ready to teach more courses on for those of us that want an excuse to go to Hawaii?

**Rick:** Yes, there are two things. I just attended the International Conference on Software Engineering, and it seemed like four out of five papers have machine learning in the title or in the abstract, and in part, it is low-hanging fruit. In part, it is that our field, no different than any other field, is kind of fashion-driven or trend-driven. But for sure, this is another tool that is now in the architect's toolbox, and so we're thinking a lot about what does it mean to create machine-learning enabled systems? How does that change the architecting process or the requirements-gathering-for-architecture process? Do we need to think about architecting in different ways? So that is definitely a technical report in my future at least.

**Suzanne:** Machine learning, one of the ways I think about it is, the learning aspect is loops of learning. That, to me, connects very strongly to architecting for evolvability because how can I allow learning to occur if I haven't architected for evolvability? That may be a connection to the machine learning that would be interesting to explore.

**Rick:** It brings up all kinds of challenges like how do we monitor our models to ensure that they are still appropriate, they are still…

**Suzanne:** Relevant.

**Rick:** They haven't drifted too far away. How do we think about modularity and modifiability for machine-learning models? Not something data scientists really were trained to do, but as these things are becoming components in big systems, you start to have to think about that. You can't treat everything as a brand-new monolith. That is what computer science as a whole has evolved away from over the last decades.

**Suzanne:** OK. So, you may be old, but you are not ready to retire, yet.

**Rick:** Yes, maybe I've got a couple more at-bats left in me.

**Suzanne:** I have always appreciated your thoughtfulness, Rick, and I know you are always thinking about how do things connect to other things? It is always a joy to talk with you about these things. You get my own, well, as I just proved, you get my own thinking going. It's like, *Well, what about this,* and *what about that?*

I hope that this is also of interest to our viewers, and I want to thank you for talking with us, today. As I said before, we will include links in the transcript to resources that we have mentioned, and a reminder to our audience that our podcasts are available virtually everywhere: SoundCloud, Stitcher, Apple Podcasts, Google Podcasts, and my favorite, the SEI's YouTube channel. If you like what you see and hear today, we would love it if you would give us a thumbs up, especially on our channel. My dad loves seeing how many…My dad is 92, and he loves seeing how many thumbs up we have on our YouTube channel. I want to thank you for joining us, Rick, and I want to thank all of our viewers for joining us today, and have a good rest of your day, wherever you are.

**Rick:** Thanks, Susie, always a pleasure.

*Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn Radio, Google Podcasts, and Apple Podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you.*