I have 2 questions for Architecture Island.

First, why is the majority of Software Architecture BAD?

Second, how do we (or is it even possible) to make the majority of Software Architecture GOOD?

In trying to capture the essence of Software Architecture, I feel like it’s a useful exercise to take a step back from the patterns, tactics, quality attributes, styles, examples, books, lectures, and everything else, and try to define what is “Good” and what is “Bad”.  Since this is a position paper, I might as well lead with a straw man position:  If a given Architecture allows the software to achieve all its goals across the lifetime of the software, then it qualifies as “Good.”  If not?  “Bad.”

Now, just to be fair, that’s pretty harsh; given the ever-increasing average lifetimes of modern systems, especially compared to the ever-shortening average employment terms as good architects find new opportunities, it’s almost impossible for the average non-clairvoyant architect to foresee all future needs of the system, and so there’s a certain amount of luck that comes in.  Still, there’s a point at which the architect can account for variation and evolution, and past that point you’re always going to want to just build a new system.

In any case, it’s also true that “Good” and “Bad” is pretty nebulous, and that’s in fact the point.  Getting to the heart of the matter of “goodness” is exactly what all of our architectural tools are meant to do; to tease out what the software needs to do for its entire lifetime, not just the now, and not even just the next.  And that’s hard!  But that’s why we define quality attribute scenarios; run ATAMs; identify patterns and tactics; and everything else.  Because if you boil everything down, to get to the essential core of Software Architecture, I suggest it comes down to answering a single question: “WHY.”  Why does the software need to scale?  Why does the software need to respond in x milliseconds?  Why will the software be attacked?  Why will the organization expect the software to run for 35 years?  Why will the hardware get bashed by a hammer?  Software Architecture thereby becomes a process of finding all the “Why’s”, answering them, and accounting for the answers in the thing you build.  Being an expert in microservices, or quality attributes, or SOA, or web servers, or networks, or Beowulf clusters will help you find the why; but it’s still the same process.  Therefore, I propose that the essence of being a software architect is finding and solving the why.  If you do that?  “Good.”  Otherwise?  “Bad.”

So, let’s take a crack at answering that first question; why is the majority bad?  I’d propose that the reason is that the majority of software engineers aren’t trained to find the “Why,” they are trained to find the “How.”  Which is great for building a solution to a specific problem, but often results in all the cruft (or “technical debt” as we like to call it) that results in “Bad.”  Even when there is an actual architect working on a design, if we’re talking about the commercial world (where most software is written), the process of finding the “Why’s” is going to be hampered by cost, schedule, and what the organization (and the architect) has done before.  It will always be cheaper and faster to use what is well understood in the institutional knowledge; rare is the architect that has both the time and schedule luxury to find the “right” solution in order to reach “Good”, when it isn’t apparent based on the needs of the project.  As such, while the architect may be able to bend the existing institutional solutions towards “Good”, they will often be too far from the ideal solution to achieve “Good”.  And, probably won’t know it until after they’ve left the project!  No wonder the majority of software architecture is “Bad.”

But, at least in my view, there is in fact hope!  The world has changed; we don’t build everything from scratch, we don’t even build everything from basic libraries.  There’s so many languages, so many frameworks, so many tools, and so many existing systems that can serve to inform the software engineers and architects of the world, that it’s easier and easier to actually achieve “Good”.  20 years ago, you were lucky to have any standardization on the web; now, with all the various OSS tools, systems, and stacks out there, WordPress rules the internet, and Apache runs it (just as one obvious example).  And we know how to use these!  And they are sufficiently understood and entrenched that even the simplest Google search will lead the most ill-informed developer will find it.  Therefore, my argument is that while building a “Good” Software Architecture will always be a challenge, most architects and engineers now have a concrete, known, and most important “Good” place to start.  They might not get there all the time, but I’m betting that eventually there’ll be enough components that the industry can lean on that we’ll tip over to 51% “Good”.  And that’s worth the effort.