Breezing My Way as a Solution Architect

A Retrospective on Skill Development and Use

Raghuraman Krishnamurthy

Successful solution architects are good interpersonal collaborators, team stimulators, experts in dealing with abstraction and technical debt, and creative thinkers who know how to look ahead. Here, Raghuraman Krishnamurthy tells his professional-skill-development story, sharing experiences gained in more than 20 years of working in multiple roles across different industries around the world. — Cesare Pautasso and Olaf Zimmermann

I’VE WORKED FOR clients across industries such as healthcare, insurance, media, pharmaceuticals, stock trading, and transportation. My experience spans consulting, application development, software maintenance, and technical presales. I’ve been lucky enough to work across multiple cultures in Europe, India, Japan, Australia, and the US.

Over the years, my work has provided the opportunity to interact with extremely successful architects. I’ve observed, almost uniformly, that accomplished architects have honed certain skills. These skills fall roughly into four areas: collaborator, stimulator (energizer or motivator), expert, and thinker (see Figure 1).

Collaborator

This area refers to the interpersonal skills of rallying people, communicating, building consensus, and being a team player.

Connect with Stakeholders

Consulting architects have a duty to help their clients move along with changing times and technological developments. The biggest challenge is articulating those developments in a language that’s appealing and relevant to the client’s business needs and IT landscape.

Example. For a pharmaceutical client, my colleagues and I wanted to explore blockchain technology. (A blockchain contains a continuously growing list of data records secured from tampering and revision.) When we spoke about blockchain technology and the wonders it can
do, the client appeared lost. The client was unable to relate the technology to its business. We did some homework and presented how, in clinical trials, blockchain technology can help increase trial report veracity. This business value argumentation resonated well with the client, and we could work on a plan for joint exploration.

Lessons learned. Be context-aware; know your audience, and tailor your messages to it in terms of style and content. Simplify and sharpen your messages adequately. Help Your Team Succeed

Architects aren’t just expert engineers, destined to work alone. An architect must bring to fruition the envisioned solution. A team of engineers will have to work together to implement that vision. As highly accomplished individuals, architects sometimes might have traces of arrogance and the desire to show that they’re right. This will create unnecessary friction.

Example. We once had a team of 20 developers working under the direction of an architect in a development project. In his zeal for excellence, the architect would arrogate to himself all decisions and was extremely opinionated. The team couldn’t work together smoothly.

So, we had coaching sessions and helped the architect understand his larger role. Identifying, encouraging, and mentoring budding architects and technical leads are important responsibilities of an architect. We gently pointed out that the architect must work for team success and be a team player.

Lessons learned. If you face volatile requirements, consider using abstraction techniques to ensure a timely and successful start; then, iterate toward perfection. Pragmatically apply lean principles and agile practices.

Be Prepared to Learn and Unlearn

Time wears away established norms, and what’s fanciful today can become a commodity tomorrow. Decades back, hardware was considered an insurmountable constraint. Changes to network configurations were tortuous. Conversations on performance tuning would center on disk I/O, CPU, and RAM limita-

Stimulator

This area concerns being able to move forward in any situation and being self-motivated.

Master the Art of Beginning

To architect enduring solutions, you should have a full appreciation of the requirements. Clarity on the needs is hard to obtain at a project’s start. The challenges are how to make a beginning without always having to start from scratch and how to avoid analysis paralysis.

Example. While working for a large insurance client, we faced a maze of requirements. Many rules existed for processing claims on policies that had several riders; under certain conditions, the rules would change wildly. Getting convergence on those rules was a never-ending task.

We abstracted the problem to a higher level and conceived the solution in five building blocks. In this case, we had abstractions for intake, quality check, processing, exceptions, and communication. This approach helped us begin the solution and then refine it iteratively.

Lessons learned. If you face volatile requirements, consider using abstraction techniques to ensure a timely and successful start; then, iterate toward perfection. Pragmatically apply lean principles and agile practices.

Be Prepared to Learn and Unlearn

Time wears away established norms, and what’s fanciful today can become a commodity tomorrow. Decades back, hardware was considered an insurmountable constraint. Changes to network configurations were tortuous. Conversations on performance tuning would center on disk I/O, CPU, and RAM limita-
tions. These limitations are no longer that insurmountable owing to virtualization and cloud computing. An architect must be prepared to acquire new knowledge, which might require shaking off established understanding and assumptions.

**Example.** I constantly learn by taking courses—for instance, from edX and Coursera. For example, I completed a Stanford University course on machine learning that helped me immensely in a consulting engagement. A major pharmaceutical company engaged us to examine how it could use machine-learning technology. I was able to work effectively and suggest potential usage areas such as sales prediction modeling (using internal data and external factors such as the weather and social media) and claims data analysis to discover unknown correlations.

Taking certificate courses for architects, membership in professional bodies such as IEEE or ACM, taking up mentoring positions, academic involvement, authoring publications, working with junior practitioners, and contributing to open source projects are all good antidotes against CV obsolescence.1

**Lessons learned.** Constant learning, applying new understandings, and being open to letting go of earlier beliefs are the keys to being successful and staying relevant in the long run.7

**Expert**

This area refers to essential trade skills.

**See the Forest and the Trees**

Being able to zoom in or out as required is important. Being extremely detail oriented yet having an abstraction capability is perhaps the distinguishing quality of a successful architect. It should be second nature to architects to possess a comprehensive understanding of the expected business capabilities, the solution architecture, how the business capabilities will be delivered through various architectural components, and how teams are working together to bring about the business capabilities. You can build this understanding by viewing the system from several perspectives (synthesis and analysis).

**Example.** I once was deputized to take over a major European airline’s cargo-handling system from another partner company. Our company was contracted to keep the system up and running while doing development in parallel. Inspired by Philippe Kruchten’s 4+1 view,8 we developed our own representation to capture a broader contextual understanding. Simultaneously, we analyzed the implementation layer by layer to slowly build a ground-up understanding of how logic was implemented. We had to skilfully navigate through the system and acquire knowledge by analyzing support tickets, interviewing the experts, and conversing with the user community. Every interaction turned out to be a veritable gold mine for filling gaps in knowledge. By persevering, we built both “forest” and “tree” knowledge that helped us serve the client well.

**Lessons learned.** Be passionate and possess an insatiable hunger for knowledge. Look for creative ways to acquire knowledge, keeping in
mind that both breadth and depth are important; everything must be meaningfully linked.9

Deal with Technical Debt
Technical debt accumulates in any large project. The reasons are myriad: shortcuts, fast-tracked development, well-intended but never-realized cleanup plans, varied skill levels on teams, and so on. It’s impossible to have clean software because time will always bring along deterioration.

It’s important to visualize the technical debt, to be better able to deal with it. Architects must understand the debt and its impact and should quantify it to decide the best way to deal with it. In extreme situations, architects might have to junk everything and start afresh.

Example. I was taking over a medical-device manufacturing control project that had just moved into production. Owing to the extremely complicated interfaces and shop floor settings, much of the testing occurred in simulated environments. When the software moved to production, it started cracking in multiple places: performance was poor, reporting was unreliable, audit logs disappeared, failure notifications were inconsistent, and so on. A review of the architecture, implementation, and testing revealed deeper problems, and we recommended significant architectural changes in the project. Predictably, the client was unhappy but reluctantly agreed. With focused release cycles in a short time frame, we delivered the application and were able to win back the client’s confidence.

Things aren’t always that drastic. For instance, my team was engaged in another project with four Oracle data transfer technologies: Oracle Streams, Oracle Advanced Queuing, Change Data Capture, and materialized views. We were unclear as to why the project used so many technologies, and we wanted to standardize to fewer technologies for simplicity. On further investigation, we found that the developers’ lack of skills, coupled with pressure from integration partners, had led the team to gradually introduce multiple technologies. We decided to live with this technical debt because it could be tolerated.

Lessons learned. Some technical debt will always be there. Know your debt, and have different strategies to deal with it. Cultivate judgment on what you must deal with and what you can tolerate.10 Follow a risk- and cost-driven approach to time management and prioritization.11

Challenge the Status Quo
An essential skill of an architect is the passion to bring simplicity and elegance. Software systems evolve over the years and pass through many caring hands. Technological advancements, changes in business processes, regulatory drivers, and new business partnerships might sometimes necessitate relooking at fundamental constructs. When you’re looking at a system, it’s important to be inquisitive to identify any dead weight and convoluted implementations. Clients often are immersed in their systems and processes, and over a long period of time, their understanding can become somewhat hidebound. Some probing, foundational questions can help shake off any incorrect notions.

Example. Years ago, while working for a major telecommunications provider, I realized how assumptions morph to almost unshakeable knowledge over time. The call records were being sent to a downstream billing system through shell scripts. They were accompanied by a database write in the call record system—ostensibly for reporting purposes. A few years after the implementation, the telecom operator and billing-services provider forged a new business collaboration. Integration between the systems now used APIs with their own metering rules. The reporting system got retired, but the database write continued. When we looked at the system landscape, we could get no convincing explanation why the database write was needed. We faced resistance to change because the system “was all working” and people had moved on to other roles. It was assumed that the database write existed for some reason.

Thinker
This refers to thinking creatively, staying focused, and not being overwhelmed by the pressures of the day.

Do your homework and challenge the status quo if you have convincing reasons.
We investigated the matter, articulated why the database write was no longer required, and presented the benefits in terms of performance, resource optimization, and simplification. With patience and continued effort, we convinced the client and could remove the obsolete connector from the architecture and code.

**Lessons learned.** Do your homework and challenge the status quo, if you have convincing reasons. Cultivate the qualities of being patient and possessing drive. Be assertive yet diplomatic.

**Don’t View the Future as an Extension of the Present**

When architecting solutions, we need to deal with a vexing dichotomy: satisfy for the present and provision for the future. Nothing is immune to change, and systems must evolve. Even when general industry trends aren’t explicitly articulated as requirements, an architect must be aware of and anticipate them and provide reasonable preliminary accommodations. Strong current trends include accessibility from mobile devices, service orientation, federated authentication, and cloud hosting. The design and implementation should be modular, scalable, and flexible.

**Example.** I was building system-to-system integration for a regulatory filing application for the US Food and Drug Administration (FDA). The system was architected for documentary submissions through user interfaces. Later, the FDA allowed submissions through its Electronic Submissions Gateway. To enable this, we had to create a service layer, which took considerable effort. Anticipating such needs would have helped us architect with a clearer separation of concerns.

**Lessons learned.** Learn to anticipate reasonable future needs and provision for them, which is a critical balancing act.$^{11,14}$

Enterprises are rapidly becoming more software-centric, and consumers are becoming increasingly comfortable and experienced with technology. Software development is moving toward agile and DevOps. The confluence of all these present interesting and challenging opportunities for software architects. By honing the skills I just described, you can enjoy your architectural career and achieve success.$^{10}$

**References**


RAGHURAMAN KRISHNAMURTHY is the senior director of technology at Cognizant Technology Solutions. Contact him at raghuraman.krishnamurthy2@cognizant.com.