EXECUTIVE UPDATE
Business & Enterprise Architecture, Vol. 24, No. 7

CUTTER CONSORTIUM
Access to the Experts

The Missing Answer of "Why Architects?"

by Pierfranco Ferronato

Software development may appear easy on the surface these days with myriad productivity tools, frameworks, and code reuse available, but software release is not the end of all efforts but rather the beginning. And that's why solution architects — with their forward-looking manner of problem solving — remain vital in the software development lifecycle. In this *Executive Update*, we examine the key question of "Why architects?" and explain that the role of solution architect, who designs for change, should not be diminished; in fact, it should be welcomed and expanded.

So what role does a solution architect still play, and why should we continue to involve architects in an IT solution?

Developing software is easy. With the right development environment and framework, productivity is typically high. Moreover, there are myriad productivity tools to streamline the process for each tier of a software system; for example, development effort is often not required for code profiling. Software developers simply can deploy a cross-hardware and cross-operating-system software solution "out of the box" and use a responsive graphical interface without much effort. Creating a database; implementing single sign-on, multifactor authentication, and authorizations; or sending messages over a network is also easy. It's like reusing/configuring existing "boxes" — to the point that talking about rapid application development, or RAD, is obsolete; it's an acronym not even known to developers nowadays.

Furthermore, reuse can also occur at a functional level, not just an infrastructure level; there are entire repositories available of free and open source solutions (e.g., customer relationship management, enterprise resource planning, e-commerce, business process management, agent frameworks, and reactive frameworks). Plus, code reuse often happens at the cloud level (e.g., software as a service, infrastructure as a service). Cloud providers offer a vast marketplace of solutions via virtual machines and container technology. In just a few seconds, a service is up and running. Of course, all these readily available systems still require a skilled and proficient team of developers; it does not happen out of the box. Software development — through any means — takes time, hard labor, and commitment.

So what role does a solution architect still play, and why should we continue to involve architects in an IT solution? As this *Update* reveals, we should not diminish the role of the solution architect in the world of today's computer science — if anything, we should expand it.

The *Executive Update* is a publication of Cutter Consortium's *Business & Enterprise Architecture* practice. ISSN: 2470-0894.

Architects: Designers for Change

Wikipedia defines "software architect" as "a software development expert who makes high-level design choices and tries to enforce technical standards, including software coding standards, tools, and platforms," while the IT architect association IASA says that architecture "is the art and science of designing and delivering valuable technology strategy." The Open Group, a global software development consortium, opines that an architect helps in defining the architecture, which is "the system's fundamental organization, embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution," according to ISO/IEC 42010:2007.

But none of these definitions gets to the point of why we need someone to reinforce technical standards and tools.

To create an effective architecture — and this doesn't mean realizing nice drawings — the solution architect must go to battle every day with developers, business owners, stakeholders, and product owners.

Without a goal, an architect is merely about leading an activity around ideals that cannot be evaluated or quantified. How can we tell if the architect performed well in the end? Did the architect create an optimal architecture by enforcing naive technical standards? Let's say that an integration pattern and a communications standard were adopted — as the architect requested — but what was the true improvement, or gain? Why was adopting such a standard a good thing, or wasn't it? Answering these types of questions does not lead to obvious answers, and if nobody believes the architect made a true impact, who would even bother the architect about anything in other projects? To create an effective architecture — and this doesn't mean realizing nice drawings — the solution architect must go to battle every day with developers, business owners, stakeholders, and product owners. The reality, unfortunately, is that the architect is often set apart in a corner with some fancy diagrams and no audience.

Let me firmly reiterate my earlier points: creating a software solution is easy; it's very easy — easy, fast, and cheap, in fact — to satisfy the up-front, cast-in-stone requirements of stakeholders.

Given this, software teams assume, apparently, that an architect is not needed. But it's vital to keep in mind that the release of the solution is not the end of the effort, but only the beginning. Indeed, the "software system" is a living creature that begins its life upon release. It is not a statue after go-live. Thus, development is not just about fixing bugs and regular maintenance. As the business evolves, the system will require changes. New rules, new policies, and new information will be requested. It is no coincidence, then, that in order to manage the continuous changes in requirements, the IT community has conceived the software lifecycle, which makes continuous delivery and integration a prime citizen, banning the waterfall approach into no-man's land.

Requirements are not cast in stone, but on the water's edge.

My push to prove the worthiness of the architect role is as valid today as if it was 40 years ago, if not more.

The reasoning behind my discussion was valid as far back as the 1980s. My push to prove the worthiness of the architect role is as valid today as it was 40 years ago, if not more. The technology stack nowadays is much taller; the levers used to architect a solution have multiplied. It's not only about a single coding language, data design, or database schema; many skills are required, even some psychology and political savvy. Nevertheless, the goal is the same: designing for change.

Evolving Requirements

So, we have assessed that software is a living thing that is continuously released and updated and thus follows the model of a continuous flow. During the software lifecycle, marketing invents new campaigns, new sales models spring up, and management defines new business processes or performs a thorough review across the organization. The point here is that it is tough to create a solution that scales and quickly adapts to evolving requirements, both functional and nonfunctional. If requirements were cast in stone when collected from users, then, yes, an architect would not be needed. But software often needs to be updated in days, if not a few hours, without disruptions, and without introducing regression errors, all while maintaining low costs, order, and discipline. This is where architecture makes its mark and supports change.

Designing with Change Outlook

The professional life of architects is tough, often dealing with tasks typically not performed in everyday non-IT life. They need to tackle both the existing, tangible problems along with future, possibly unforeseen challenges. Thus, architects always must support a "future state," a situation not yet in existence that users or stakeholders can't predict if not stressed with the proper questions. Architects must look "beyond the hill" and impose superstructures and patterns that may appear unnecessary today but are useful when things ultimately change.

Architects are closer to the business than to the technology; they know that the former affects the latter.

This requires architects to have a "vision" beyond the outlook of their stakeholders. With such a long-range view, architects can set standards and models based on future conditions outlined in a predictive model that depends on the underlying business model supported by the system itself. Architects are closer to the business than to the technology; they know that the former affects the latter. Therefore, architects are professionally deviated. When listening to a requirement, or reading an analysis document, architects care little about how to fulfill the need. They know, essentially, that it's easy; a good developer can write the code. Instead, they worry about how that specification is fated to change. Architects have a "paranoia" for abstraction, a need to find solutions that solve a class of problems, not just the instance of the problem itself.

Imagining Scenarios

If someone presents me with the problem of moving a car from A to B, I first think: What type of car is it? Does this involve more than one type of car? If so, what are the other types? Can the car in question move by itself? Do you need to move more cars to make this happen? Are motorcycles involved?

Do you see where I am going? Generally speaking, there's no real problem when moving a car from A to B; hence, there's no need for an architect if you stick to that main requirement and do not change anything. But this is never true in business. Inevitably, change will happen — if you want to keep your business alive.

The business is dynamic by nature and so must be the software. Of course, architect solutions to problems must be "reasonably" scalable; obviously, supporting a case of moving ships or planes requires an entirely different business model. Thus, in that scenario, the architect absolutely needs to break the solution down into sub-problems. However, the underlying rational mechanism for any problem remains: it is about looking into the future. The goal is to implement scalable solutions that do not solve a single problem, but rather a class of problems to which it belongs: a meta-solution.

Architects are closer to the business than to the technology; they know that the former affects the latter.

Now, if I want to support a future scenario of moving motorcycles, I have to prepare[1] — this is the keyword of architects. I would need specific hooks on the floor for pulling down ropes attached to motorcycles' handlebars and shocks; these are different than the hooks needed for cars. Alternately, looking ahead, if I want to support transporting more than one car, I will need a trailer ready and certified for multiples cars, so as not to waste any time during the transport operation. Yet, when it comes to moving a single car, tow hooks might be completely useless. Maybe the car just needs to be pushed. It is at this point where the architect, with multiple scenarios on top of mind, might have conflicts with other members of the team who are focused on the actual problem solving.

Facing Obstacles

"But it works anyway," I'm sometimes challenged. The questions then go on and on: Why on earth do we have to make a "façade," or a reverse proxy? Why an event broker instead of making a direct call? Why break a component into several parts? Why use a UNI/ISO protocol? Why a restful standard for API? The developers then grumble that they have to write more code, which makes it more complicated. The team accuses the architect of not understanding, of being attached to the "aesthetic beauty" of a solution; they

[1]As Wikipedia explains: "the word 'preparation' comes from late Middle English (1350–1400) via Old French from Latin *praeparationem/praeparatio* … 'a making ready.' It is from a past participle stem of *praeparāre* meaning to prepare; from *prae* 'before' + *parare* 'make ready.' "

usually say that architects are not pragmatic and often just a "thorn in their side" for the duration of the first delivery.

Ultimately, the struggle is unequal in a battle between one who thinks about hypothetical future states of a situation and those that merely measure time and cost to realize something of immediate and measurable value.

These arguments and reasonings are taken to the upper floors of management. Developers complain to the project leader, who in turn is supported by the business manager, who says that "there is no time and the clock is ticking." The architect is reassured that the workaround will be registered as technical debt and will be done after the release. The architect, however, will be alone defending the maintenance of standards and the architectural style. The architect will not find fertile ground for support and will be left pointing out that short-term decisions will be at the expense of future requests, a prediction based on the architect's past experience, along with the volatility of requirements in that specific business sector. The architect's naysayers then yell, "He is a wizard; he pretends to predict the future!"

Ultimately, the struggle is unequal in a battle between one who thinks about hypothetical future states of a situation and those that merely measure time and cost to realize something of immediate and measurable value. Without a strong mandate within the software factory, a delegation, or formal authority in the checklist, the architect risks being neglected. In my experience, the architect is taken "out of the game" after a go-live, as if not needed anymore. "Thanks for taking us to the successful go-live," the architect is told. "Now it's on Operation and Support."

But a system that is not maintained or aligned with an architectural style is doomed to chaos and rapid obsolescence. This is not a critique of business, project leaders, or project managers, and users are not to blame. It is merely the nature of things, a characteristic of the physics of matter, of quantum mechanics: it is called "entropy." It's an increasing function of the probability that a system is in a given macroscopic state, spontaneously evolving toward configurations with greater entropy, which are at a lower degree of order. It is, therefore, normal that a system without energy injection tends "spontaneously" toward less order and, thus, toward disorder, often interpreted as the degree of disorder or randomness in the system. With clutter, a software system tends to be clumsy; changes

required by new, returned requirements end up having cascading effects on other areas of the code, increasing the likelihood of errors in other seemingly separate logical contexts. Consequently, code fixes "mysteriously" take a lot of time, while at a first glance may seem trivial.

Parting Thoughts

Sadly, because of the nature of an architect who designs for change, an architect's accomplishments typically are appreciated only years later, after being long gone or assigned to other projects or other organizations. Architecture provides the structures that naturally support functional evolutions, keeping entropy under control. Certainly, there will be cases where architecture revision (or extension) will be needed to introduce a new pattern or mechanism not considered up until that point. This is why the architect should be involved again after release into production; however, in my experience, the architect is often pushed out after a successful go-live. Just like in civil engineering: the building is finished; the job is done.

Architecture provides the structures that naturally support functional evolutions, keeping entropy under control.

The IT architect's role is still very recent in the history of labor and still suffers from a mindset bias. Comparing IT architects with those in civil engineering is misleading; there is a key difference. A civil architect must accurately plan to avoid redoing things, while an IT architect must allow for modifications to almost everything. Too often, architects are perceived as only serving to create the software system. Nobody expects a bridge to move, stretch, increase in lanes, or double in its expected traffic load while still letting a car pass through it; the requirements are essentially static. All tangible things are like this; moving a pillar, for example, has a cost that is an order of magnitude greater compared to building it. Moving a window in a house is far more expensive than building it. Thus, the civil architect plans in advance to avoid redoing things; otherwise, the costs would be exorbitant and doom the project to failure. The IT architect, however, must instead apply strategy to allow for the modification of almost everything, but since the software is … "soft," it's cheaper to modify code rather than to write it, so why bother hiring an architect? … Or maybe it isn't.

About the Author

Pierfranco Ferronato is an innovative and visionary lead solutions architect with long-term vision and extensive experience within infrastructure, applications, and cloud computing solutions. He has a broad understanding of business and customer requirements with proven talent in design and development of cost-effective solutions in answer to complex opportunities. Dr. Ferronato is an enthusiastic advocate for innovative technological solutions with a consultative approach tailored appropriately to the listener. He has been published widely by the IEEE in architecture methods and solutions. Dr. Ferronato has served as an enterprise architect, senior lead architect for Huawei R&D Dublin, principal architect for a major telco, technical coordinator of EU-funded projects of about €15 million, CTO, and solution architect for various international organizations. Currently, he is a Senior Application Architect for the European Food Safety Authority (EFSA). Dr. Ferronato holds a master's degree in computer science from the University of Udine, Italy, and has TOGAF and ITIL certifications. He can be reached at pierfranco@ferronato.eu.

About Cutter Consortium

Cutter Consortium is a unique, global business technology advisory firm dedicated to helping organizations leverage emerging technologies and the latest business management thinking to achieve competitive advantage and mission success. Through its research, training, executive education, and consulting, Cutter Consortium enables digital transformation.

Cutter Consortium helps clients address the spectrum of challenges technology change brings — from disruption of business models and the sustainable innovation, change management, and leadership a new order demands, to the creation, implementation, and optimization of software and systems that power newly holistic enterprise and business unit strategies.

Cutter Consortium pushes the thinking in the field by fostering debate and collaboration among its global community of thought leaders. Coupled with its famously objective "no ties to vendors" policy, Cutter Consortium's Access to the Experts approach delivers cutting-edge, objective information and innovative solutions to its clients worldwide.

For more information, visit www.cutter.com or call us at +1 781 648 8700.