Common Misconceptions about Software Architecture

by Philippe Kruchten
Rational Fellow
Rational Software Canada

References to architecture are everywhere: in every article, in every ad. And we take this word for granted. We all seem to understand what it means. But there isn't any well-accepted definition of software architecture. Are we all understanding the same thing? We gladly accept that software architecture is the design, the structure, or the infrastructure. Many ideas are floating around concerning why and how you design or acquire an architecture and who does it. In this article I review some of these accepted ideas and show why, in my opinion, they may be misconceptions.

"Architecture is design."

Yes, architecture is design. It is about making the difficult choices on how the system will be implemented. It is not just the "what."

But not all design is architecture. We see this word applied more and more frequently to any form and aspect of design. A few years ago, Mary Shaw pleaded: "Do not dilute the meaning of the term architecture by applying it to everything in sight." Unfortunately things have become worse, not better.

Architecture is one aspect of the design, focusing on the major elements -- the elements that are structurally important, but also those that have a more lasting impact on the performance, reliability, cost, and adaptability of the system. Architecting is choosing the small set of mechanisms, patterns, and styles that are going to permeate the rest of the design and give it its integrity. Architecture is the tool that allows us to master complexity. It cannot be the whole design. It has to limit itself to a certain level of abstraction but still be concrete enough to draw definite conclusions. It is not just "high-level design."

What shall the architect focus on, then? There is no universal answer. For any given project, a decision needs to be made about what is architecturally significant so that we can draw that thin and elusive line between architecture and the rest of the design activities.

"Architecture is infrastructure."

Yes, the infrastructure is an integral and important part of the architecture: It is the foundation. Choices of platform, operating systems, middleware, database, and so on, are major architectural choices.

But there is far more to architecture than just the infrastructure. The architects have to consider the whole system, including all applications; otherwise an overly narrow view of what architecture is may lead to a very nice infrastructure, but the wrong infrastructure for the problem at hand. Time and time again, I run into this in organizations in which an architecture team is working solely on infrastructure, largely ignorant of the problem domain and the application software -- which they consider to be outside of the architecture. "Oh, you mean the application. That's what the people in the other building do."

"Architecture is [insert favorite technology here]."

"The network is the architecture. The database is the architecture. The transaction server is the architecture. The GUI is the architecture. CORBA is the architecture. This standard is the architecture..." This is a special case of the previous point. Yes, many of these aspects are part of the architecture, but the architecture cannot be restricted to one aspect only.

Architecture is more than just a "technology watch," but I see many organizations in which the major role of the software architect seems to be experimenting with interesting new technologies. Often this is also the consequence of having architects who all come from one single specialty: for example, an architecture team comprised solely of data engineers. I was told recently: "We do not need anybody to work on architecture; our company has standardized on three-tier client-server architecture."

"Architecture is the work of a single architect."

"A great architecture is the work of a single architect. " Fred Brooks wrote this in 1975.[1] Granted, there are some great examples of this: F. Brooks himself, G. Amdahl, or A. Kay of SmallTalk fame.[2] But in practice, geniuses are rare, and in many organizations good architectures are most often the work of a small group of people working as a team.

The architecture team acts as a team as defined by Katzenbach and Smith[3]:

> A team is a small number of people with complementary skills who are committed to a common purpose, performance goals,

and approach for which they hold themselves mutually accountable.

So the architecture team is not a committee, meeting every Tuesday at 15:00; it is not a problem clearinghouse; it is not an ivory tower, a resting place for successful but tired designers.

Especially for complex systems, a well-chosen architecture team is needed to bring the right mix of domain and software engineering experience, and the right mix of design specialties (hence minimizing the [my favorite architecture] effect mentioned above). The architecture team needs significant flexibility in composition and structure. Some of the architects may become the architects of subsystems, once those have been defined. However, it is important to have a clearly designated team leader: a lead software architect who can drive the effort, arbitrate, resolve conflicts, and bring timely closure to project tasks.

"Architecture is flat."

When I ask to see an architectural description, I often notice that people have tried very hard to make it flat -- two- or even one-dimensional. They attempted to convey many different aspects of architecture using only one kind of concept, one family of components and connectors. They end up trying to show the many complex concerns of multiple stakeholders using one kind of diagram or blueprint. The description, however, is incomplete. A flat language cannot capture or describe the intricacies of a complex system. And I have the same concerns about some architectural description languages.

Architecture is a complex beast; it is many things to many different stakeholders. Using a single blueprint to represent architecture results in an unintelligible semantic mess. Like building architects who have floor plans, elevations, electrical cabling diagrams, and so on, we need multiple blueprints to address different concerns, and to express the separate but interdependent structures that exist in an architecture. As a solution, I had proposed an architectural representation based on four main views plus one,[4] and others have proposed similar approaches, leading recently to an IEEE Standard.[5]

"Architecture is structure."

Again, the answer is "yes, but no." Yes, architecture must describe the structure of the system, and there are multiple structures intertwined in the architecture as we just saw. But no, there is more to architecture than just hierarchical decomposition, layering, or pipelining.

Architecture must also deal with dynamic issues that are beyond (or across) structure and only vaguely visible at the interfaces: the flows of messages or events, the protocols, the state of machines, the creation of threads.

Architecture must also address the "fit": How will the ultimate system fit in its contexts? This includes both the operational context (Does it meet the needs of its users?) and the developmental context (Is it easy to build? Is it simply feasible with the resources at hand?). These issues go beyond mere structure, although some try to reduce them to that.

"System architecture precedes software architecture."

Yes, you need to define and structure the system as a whole before you can say a thing about software. But far too often I still see software-intensive systems being first completely designed by system engineers, and when all major system-level decisions have been made, the system engineers open the door and say: "Now you software types, you can come in. We'll show you where the computing nodes are, and you can sprinkle your magic."

This approach usually results in stovepipe systems: islands of software isolated on various computers. A large amount of effort is then dedicated to defining the interfaces between the various software subsystems. Most opportunity for software reuse across the system is nipped in the bud because the software architecture was not developed in conjunction with the system architecture, and the final system is not resilient to changes in the underlying hardware. And when it is discovered that an error or miscalculation was made at the system level, it becomes the job of the software types to fix it in software. This problem is even worse when the development of different software subsystems is contracted to different organizations. In long-term development efforts, it is likely that the underlying system will change during software development -- the hardware that was bid at the beginning of a project may not be even manufactured at the time of its delivery.

For software-intensive systems, system architecture and software architecture must proceed simultaneously. Their activities must be interleaved, feeding each other with solutions, opportunities, and constraints. The architectures must be almost indistinguishable: The system architecture is -- and includes -- the software architecture.

"Architecture cannot be measured or validated."

Architecture is not just a whiteboard exercise that results in a few interconnected boxes and is then labeled a high-level design. There are many aspects you can validate by inspection, systematic analysis, simulation, or modelization. And I am all for using them appropriately. But in the end, all these techniques may only point to something in the proposed architecture that *may not* work. You can only inspect, model, and analyze what is known and described; architectural flaws most often come from the unknown.

I have come to believe that to validate that an architecture *will* work, you have to implement it and try it. Winston Royce and Walker Royce wrote:[6]

> ... thus software architects have no irrefutable first principles; without available theory, their starting point must be some form of experimentation. Experimentation via simulation is a

possibility, but it rarely works for software, primarily because high-fidelity, easy to build simulations are only possible to build if the underlying physics is known or a mathematical model can be assumed -- exactly what is missing in the first place. The remaining experimental alternative is prototyping, where the word *prototype* is used with a very narrow, specific meaning; i.e., building something with the same design as is intended for the final deliverable product.

Building a skeletal architecture in early iterations is a method of architecture validation with many benefits. It helps us to:

- Mitigate technical risks by trying proposed solutions early in the design process.
- Progress, in both the problem domain and the solution domain.
- Reduce integration risks.
- Jump-start the testing effort by providing something to integrate and test early in the development lifecycle.
- Gain experience with tools and people so that both operate smoothly when we need them later.
- Set up appropriate expectations in terms of functionality, performance, completion date, and cost, so that there are no surprises late in the project.

A core tenet of the Rational Unified Process is an early focus on the design and validation of a baseline architecture in its elaboration phase.

"Architecture is a science."

Not yet. Scientific, analytical methods are hard to apply to anything that is not ridiculously small, as the Royces said. There is no real proof that the architecture will work, other than prototyping for some aspects, because there are few quantitative, objective criteria.

Usually, time is of the essence when designing system architectures. The architects have no latitude to systematically study all possible solution paths and their combinations in order to come up with the optimal solution; they must rapidly make decisions to allow work to proceed. There is no point in coming up with the ideal solution after the battle is lost. I often describe the life of a software architect as a long and rapid succession of suboptimal design decisions taken partly in the dark. It is not a static function that we are optimizing anyway. Neither the constraints nor any parts of the problem are static enough for long enough to approach anything "optimal." Architecture is not about the optimal, or ideal; it is about the adequate, or satisfactory.

The nasty consequence is that when one looks at the architecture of the system after it has been implemented, it is far too easy to criticize every design decision that was made. With history on your side, you know much more than what the architects knew at the time they were making the decisions, both about technology and the requirements. "Why didn't you use CORBA? Or this tool?" the *ex post facto* critics will ask, but at the time the decision needed to be made, not enough information existed to make such a choice. In order for the project to progress, plans must be laid, decisions must be made, and the team must move on.

Will architecture ever become a science? Not very rapidly. It needs to first reach the level of engineering. I know computer scientists who will cringe when reading this.

"Architecture is an art."

Whoa. Let's not fool ourselves. Some architects may like to portray themselves as magicians: "Hire me. I'll look at your project, retreat onto my mountain, levitate for a while in trance, then come down with the solution." The artistic, creative part of software architecture is usually very small. Most of what architects do is copy solutions that they know worked in other similar circumstances, and assemble them in different forms and combinations, with modest incremental improvements.

It is possible to describe an architectural process that has precise steps and prescribed artifacts, and that takes advantage of heuristics and patterns that are starting to be better understood. We are attempting to do this in a modest way in the Rational Unified Process.[7] But our description also emphasizes the importance of having a team of architects with a broad spectrum of experience.

"These are the top ten misconceptions."

Maybe the biggest misconception of all is to think that we know enough today about software architecture for an individual like me to stand up and declare that these are misconceptions! My certainties may be misconceptions in your eyes, and vice versa. Whatever you believe, however, at least consider these ten topics as important issues that deserve some attention, or potential traps into which I hope you will not fall.

Acknowledgments

I would like to thanks my friends and colleagues Grady Booch, Dean Leffingwell, Kurt Bittner, Rich Hilliard, Walker Royce, Rick Kazman, and Jaswinder Madhur for their insightful reviews of an earlier draft of this article.

References

F. P. Brooks, Jr. *The Mythical Man-Month*. Reading MA: Addison-Wesley, 1975.

D. Shasha and C. Lazere, *Out of Their Minds--The Lives and Discoveries of 15 Great Computer Scientists*. New York: Copernicus, 1995.

J. Katzenbach and D. Smith, *The Wisdom of Teams*. New York: HarperBusiness, 1993.

P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, 12 (6), November 1995, IEEE, pp. 42-50.

IEEE Standard 1471:2000, Recommended Practice on Architectural Description.

W. E. Royce and W. Royce, "Software Architecture: Integrating Process and Technology," *Quest*, 14 (1), 1991, TRW, pp. 2-15.

Rational Software Corp. *The Rational Unified Process*, version 2001, 2001.

E. Rechtin and Mark Maier, *The Art of Systems Architecting*, CRC Books.

---

[1] F. P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley, Reading MA, 1975.

[2] D. Shasha and C. Lazere, *Out of Their Minds--The Lives and Discoveries of 15 Great Computer Scientists*, Copernicus, New York, 1995.

[3] J. Katzenbach and D. Smith, *The Wisdom of Teams*. New York: HarperBusiness, 1993.

[4] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, 12 (6), November 1995, IEEE, pp. 42-50.

[5] IEEE Standard 1471:2000, Recommended Practice on Architectural Description.

[6] W. E. Royce, and W. Royce, "Software Architecture: Integrating Process and Technology," *Quest*, 14 (1), 1991, TRW, pp. 2-15.

[7] Rational Software Corp. *The Rational Unified Process*, version 2001, 2001.
