**RC24924 (W0912-126) December 23, 2009**
**Computer Science**

# IBM Research Report

## On Architecture: Achitecture as a Shared Hallucination

**Grady Booch**
IBM Research Division
6533 West Prentice Avenue
Littleton, CO 80123

# Architecture as a Shared Hallucination

*Grady Booch*

In Jane Wagner's play *The Search for Signs of Intelligent Life in the Universe*, Lilly Tomlin's character Trudy remarks "What is reality, anyway? A collective hunch." She goes on to suggest "reality was once a primitive method of crowd control."

So it is with the architecture of a software-intensive system.

Architecture is just a collective hunch, a shared hallucination, an assertion by a set of stakeholders on the nature of their observable world, be it a world that is or a world as they wish it to be. Architecture therefore serves as a means of anchoring an extended set of stakeholders to a common vision of that world, a vision around which they may rally, to which they are led, and for which they work collectively to make manifest.

When I say that architecture is a shared hallucination, what I mean is that architecture-as-an-artifact is a naming of the mutually agreed upon set of design decisions that shape a software-intensive system. While architecture is just an abstraction of reality, architecture-as-an-artifact is a declaration of that shared reality. In this manner, that shared hallucination represents a common vision among a set of stakeholders as observed simultaneously through several different points of view and represented by a set of interlocking models.

Now, to be clear, a model is just an abstraction of reality; a model is not reality. Although he was talking about statistical models, George Box has quite properly observed that "all models are wrong, but some of them are useful." In other words, using Scott McCloud's phrase, we use models to achieve amplification through simplification. A useful model focuses on a particular concept to raise it above the cacophony of the complexity that surrounds it.

Building useful abstractions is hard for we require our abstractions to have degrees of freedom while at the same time being unambiguous. For example, if I am trying to reason about the mix of services I want to expose in a given enterprise, in one view I may wish to consider the physical components that deliver up those services and where they are deployed in a given network; in another view, I may wish to consider the granularity of those services from a functional perspective. In both cases, I'll want to have a fairly unambiguous statement of the semantics of these services, yet in the former case I may abstract away the implementation of those services (to focus on their deployment) while in the latter I may abstract away the location (to focus on their functionality). This is why one view is never enough: if you try to pile all the meaning of a thing into one view, you end up with the code itself, and that complexity will obscure the more delicate yet important threads that shape the system.

As Carl Sagan noted in *Cosmos*, "the brain does more than just recollect. It compares, synthesizes, it analyses, it generates abstractions." He goes on to say that "the brain has its own language for testing the structure and consistency of the world." This is why

I'm a rabid fan of graphical models: they allow us to present an abstraction with some degrees of freedom yet without ambiguity, and then let the brain do the reasoning. As Edward Tufte notes "The point is that analytical designs are not to be decided on their convenience to the user or necessarily their readability or what psychologists or decorators think about them; rather, design architectures should be decided on how the architecture assists analytical thinking about evidence." We render our abstraction in models, and we use those models to reason about our abstractions.

When I speak of being able to reason about a model, what I mean is that the best models have two important characteristics: first, they codify a design decision in a manner that is predictive and second, that serves as the basis against which one can ask questions. For example, back to my services scenario, a good model is one that would be predictive in that, if I invoke a service, I should be able to understand what other services are triggered and thus collaborate as that service plays out. Similarly, a good model should allow me to ask questions, such as how long might I expect a service to take to respond, or where might the bottlenecks arise were I to trigger a set of services as such and such a frequency.

You have heard me say in this column that the code is the truth, but it is not the whole truth. In the common vision that an architecture-as-an-artifact provides, there are five threads that interweave:

- Architecture as a collection of significant design decisions
- Patterns as the themes
- Cross-cutting concerns as the traces
- Rationale as the back story
- Tribal memory as the human story

Thus, a description of a system's living architecture is formed from technical as well as social elements.

The first three of these threads are primarily technical in nature. The significant design decisions we assert give rise to the essential shape of the system, much like the load-bearing walls of a building or the essential mechanisms that operate within a cell. The patterns we find name the societies of abstractions that collaborate together; a given abstraction may participate in my many such patterns, and the presence of those patterns give rise to elegance and simplicity in the face of essential complexity. Cross-cutting concerns – elements such as security or performance – represent things that are architecturally significant, but yet are so deeply embedded in the very fabric of a system that we cannot reason about them unless we pop up a level of abstraction; these are things that, more so than patterns, give rise to emergent behavior.

The remaining two threads are primarily social in nature. As for rationale, this is the metaknowledge as to why we chose one decision over another; as for tribal memory, this is the institutional knowledge of who did what when and why. A body of code, even if well-commented, represents the winner of all the hundreds of thousands of small design decisions. Be it for overwhelming technical reasons, or historical ones, or hysterical ones, or even emotional ones, what was not will be lost in time over time unless something is done to preserve that metaknowledge. In the moment, it is easy to make a choice and then move on, but for decisions that are linchpins – which in mechanical engineering are important elements of support and stability – the reason they were chosen as such mainstays will be lost in the dust of the decisions that fall down over it. Later on, people will either be afraid to touch that bit of code ("it works but no one understands it, so let's leave it alone") or they will touch it accidently, and the whole system will fall down upon itself.

A system's architecture not only provides a common vision, it also provides a vision around which stakeholders may rally. This is where architectural governance fits in, in that the architecture-as-an-artifact serves as the backbone against which all other decisions are made. In this manner, a system's architecture becomes the place where shared trust is made manifest; this is the shared hallucination. But, it's more than just a place where in code we trust, it's also the place where new stakeholders can be brought to orient themselves. Consider your experience when parachuted in to a new code base: you'll spend time pouring over the code, but more often than not, you'll seek out and hold hand, someone who has that code under their nails, and ask them to give you the lay of the land. That lay of the land is the system's architecture, and the better it is made visible, the more transparent the decisions that formed it, the easier it is to trust the architecture and to efficiently grok that lay of the land.

As an artifact to which stakeholders are lead, a well-syndicated architecture serves as sort of a strange attractor, a stable intermediate form that shapes the mob of stakeholders who work on that sytem. As Steve McConnell mentioned in the book *Beautiful Teams*, it's important in any venture to have an elevating goal. As he notes, "A classic example is if you're out digging ditches, that's not very elevating or inspiring. But if you're digging ditches to protect your town that's about to be attacked by an enemy, well, that's more inspiring, even though it's the same activity." The statement of a system's architecture  can be the place where such a goal is named.

Finally, there's architecture as the place where we grow the system in controlled, steady fashion, where we make our vision manifest. From models to executables, some transformations are tedious, some transformations are noisy, some transformations are lossy, and some transformations are somewhat reversible. Those that are tedious lend themselves to model-driven development; those that are reversible lend themselves to automated mining; those that are noisy or lossy require human intervention.

As the architet van der Rohe once said, "architecture starts when you carefully put two bricks together." For software-intensive systems, architecture starts when you lay down your first line of code, thus beginning the shared hallucination.