Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

Who Needs an Architect?

Martin Fowler

Wandering down our corridor a while ago, I saw my colleague Dave Rice in a particularly grumpy mood. My brief question caused a violent statement, "We shouldn't interview anyone who has 'architect' on his resume." At first blush, this was an odd turn of phrase, because we usually introduce Dave as one of our leading architects.

The reason for his title schizophrenia is the fact that, even by our industry's standards, "architect" and "architecture" are terribly overloaded words. For many, the term "software architect" fits perfectly with the smug controlling image at the end of *Matrix Reloaded*. Yet even in firms that have the greatest contempt for that image, there's a vital role for the technical leadership that an architect such as Dave plays.

What is architecture?

When I was fretting over the title for *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002), everyone who reviewed it agreed that "architecture" belonged in the title. Yet we all felt uncomfortable defining the word. Because it was my book, I felt compelled to take a stab at defining it.

My first move was to avoid fuzziness by just letting my cynicism hang right out. In a sense, I define *architecture* as a word we use when we want to talk about design but want to puff it up to make it sound important. (Yes, you can imagine a similar phenomenon for architect.) However, as so often occurs, inside the blighted cynicism is a pinch of truth. Understanding came to me after reading a posting from Ralph Johnson on the Extreme Programming mailing list. It's so good I'll quote it all.

A previous posting said

> The RUP, working off the IEEE definition, defines architecture as "the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces."

Johnson responded:

> I was a reviewer on the IEEE standard that used that, and I argued uselessly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. So, perhaps an architecture is the highest level concept that developers have of a system in its environment. Let's forget the developers who just understand their little piece. Architecture is the highest level concept of the expert developers. What makes a component significant? It is significant because the expert developers say so.
>
> So, a better definition would be "In most successful software projects, the expert developers working on that project have a shared understanding of the

system design. This shared understanding is called 'architecture.' This understanding includes how the system is divided into components and how the components interact through interfaces. These components are usually composed of smaller components, but the architecture only includes the components and interfaces that are understood by all the developers."

This would be a better definition because it makes clear that architecture is a social construct (well, software is too, but architecture is even more so) because it doesn't just depend on the software, but on what part of the software is considered important by group consensus.

There is another style of definition of architecture which is something like "architecture is the set of design decisions that must be made early in a project." I complain about that one, too, saying that architecture is the decisions that you wish you could get right early in a project, but that you are not necessarily more likely to get them right than any other.

Anyway, by this second definition, programming language would be part of the architecture for most projects. By the first, it wouldn't be.

Whether something is part of the architecture is entirely based on whether the developers think it is important. People who build "enterprise applications" tend to think that persistence is crucial. When they start to draw their architecture, they start with three layers. They will mention "and we use Oracle for our database and have our own persistence layer to map objects onto it." But a medical imaging application might include Oracle without it being considered part of the architecture. That is because most of the complication is in analyzing the images, not in storing them. Fetching and storing images is done by one little part of the application and most of the developers ignore it.

So, this makes it hard to tell people how to describe their architecture. "Tell us what is important." Architecture is about the important stuff. Whatever that is.

The architect's role

So if architecture is the important stuff, then the architect is the person (or people) who worries about the important stuff. And here we get to the essence of the difference between the *Matrix Reloaded* species of architect and the style that Dave Rice exemplifies.

Architectus Reloadus is the person who makes all the important decisions. The architect does this because a single mind is needed to ensure a system's conceptual integrity, and perhaps because the architect doesn't think that the team members are sufficiently skilled to make those decisions. Often, such decisions must be made early on so that everyone else has a plan to follow.

Architectus Oryzus is a different kind of animal (if you can't guess, try www.nd.edu/~archives/latgramm.htm). This kind of architect must be very aware of what's going on in the project, looking out for important issues and tackling them before they become a serious problem. When I see an architect like this, the most noticeable part of the work is the intense collaboration. In the morning, the architect programs with a developer, trying to harvest some common locking code. In the afternoon, the architect participates in a requirements session, helping explain to the requirements people the technical consequences of some of their ideas in nontechnical terms—such as development costs.

In many ways, the most important activity of *Architectus Oryzus* is to mentor the development team, to raise their level so that they can take on more complex issues. Improving the development team's ability gives an architect much greater leverage than being the sole decision maker and thus running the risk of being an architectural bottleneck. This leads to the satisfying rule of thumb that an architect's value is inversely proportional to the number of decisions he or she makes.

At a recent ThoughtWorks retreat, some colleagues and I were talking about the issue of architects. I found it interesting that we quickly agreed on the nature of the job, following *Architectus Oryzus*, but we could not easily find a name. *Architectus Reloadus* is too common for us to be comfortable with "architect," and it's based on a flawed metaphor (see http://martinfowler.com/bliki/BuildingArchitect.html). Mike Two came up with the best name I've heard so far: *guide*, as in mountaineering. A guide is a more experienced and skillful team member who teaches other team members to better fend for themselves yet is always there for the really tricky stuff.

What makes a component significant? It is significant because the expert developers say so.

Getting rid of software architecture

I love writing a startling heading, and the best, like this one, have an important meaning that's not immediately obvious. Remember Johnson's secondary definition: "Architecture is the decisions that you wish you could get right early in a project." Why do people feel the need to get some things right early in the project? The answer, of course, is because they perceive those things as hard to change. So you might end up defining architecture as "things that people perceive as hard to change."

It's commonly believed that if you are building an enterprise application, you must get the database schema right early on because it's hard to change the database schema—particularly once you have gone live. On one of our projects, the database administrator, Pramod Sadalage, devised a system that let us change the database schema (and migrate the data) easily (see http://martinfowler.com/articles/evodb.html).

By doing this, he made it so that the database schema was no longer architectural. I see this as an entirely good thing because it let us better handle change.

At a fascinating talk at the XP 2002 conference (http://martinfowler.com/articles/xp2002.html), Enrico Zaninotto, an economist, analyzed the underlying thinking behind agile ideas in manufacturing and software development. One aspect I found particularly interesting was his comment that *irreversibility* was one of the prime drivers of complexity. He saw agile methods, in manufacturing and software development, as a shift that seeks to contain complexity by reducing irreversibility—as opposed to tackling other complexity drivers. I think that one of an architect's most important tasks is to remove architecture by finding ways to eliminate irreversibility in software designs.

Here's Johnson again, this time in response to an email message I sent him:

> One of the differences between building architecture and software architecture is that a lot of decisions about a building are hard to change. It is hard to go back and change your basement, though it is possible.
>
> There is no theoretical reason that anything is hard to change about software. If you pick any one aspect of software then you can make it easy to change, but we don't know how to make everything easy to change. Making something easy to change makes the overall system a little more complex, and making *everything* easy to change makes the entire system very complex. Complexity is what makes software hard to change. That, and duplication.
>
> My reservation of Aspect-Oriented Programming is that we already have fairly good techniques for separating aspects of programs, and we don't use them. I don't think the real problem will be solved by making better techniques for separating aspects. We don't know what should be the aspects that need separating, and we don't know when it is worth separating them and when it is not.
>
> Software is not limited by physics, like buildings are. It is limited by imagination, by design, by organization. In short, it is limited by properties of people, not by properties of the world. "We have met the enemy, and he is us."

Martin Fowler is the chief scientist for ThoughtWorks, and Internet systems delivery and consulting company. Contact him at fowler@acm.org.