Architecture Deployment

(SPAMMED architecture framework series)

Arnon Rotem-Gal-Oz

[This whitepaper is based on a series of blog posts that first appeared in Dr. Dobb's Portal www.ddj.com/dept/architect]

The Spammed Architecture Framework (SAF) indentifies the Deployment activity as part of the architect's work to ensure the success of a project.

The Deployment activity tries to address with the following set of issues:

- How do we know someone does not adhere to the architecture?
- What do we do once we find such a violations?
- Can we avoid architecture violations in the first place? And if yes – how?
- How involved should the architect be?
- How do we make sure "the architecture" stays relevant throughout the project?

I'll address the last issue (ensuring the architecture is relevant) first and tackle the rest (which are tightly related later)

Staying relevant

One important thing we should bare in mind is that there is no "architecture phase" per se. The normal course is for the architecture to stabilize in the first few iterations and to remain more or less stable after that (unless you get major requirements changes -- which will prompt you to re-evaluate and possibly redesign the architecture). Note that the architecture is dependant mostly on quality attributes of the system. Quality attributes tend to be more stable than functional requirements (even though, I've seen quality attributes change in significant ways, for example, in a startup I worked for, we once made a wee change in our target customers --from SMBs to Fortune 500. As can be expected, this

change really turned almost everything, except maybe the splash screen, upside down.)

We can expect the initial architecture formed in the first one or two iterations to be lacking. This is a reasonable assumption, since the initial architecture matches your initial understanding of the system (Which usually, is also partial). Unless, of course, the project is really trivial. I do find it useful to have an initial architecture phase (where you don't deliver features and concentrate on architectural decisions) -- I usually try to limit this initial phase to one, maybe two iterations (usually 1-2 months) and have it end with a working skeleton that demonstrates how all the components play together.(e.g. by implementing one thread through a use case or implementing a central user story). But, again, this does not mean the architecture is "done" -- it means we (hopefully) have a sound foundation to start building the software.

One goal of the initial architecture is to leave enough leeway to allow changes without having to throw everything and start anew. The more you are familiar with similar projects the more you can expect the architecture to be close to final one, but remember, the opposite is also true, if you are new to a project type you can expect the architecture to be far from final.

How do you keep the architecture relevant throughout the project? You keep yourself involved in the project, validate the architecture against the requirements and be ready to evolve the architecture if needed. You should strive to stabilize the architecture after the first few iterations -- to do that the architect should try to find the use cases/user stories that challenge the architecture or that are risky and prioritize them higher. Note however that the customer has precedence and the initial iterations might be focused on delivering some business value that doesn't challenge you architecture; (which is why I recommend giving at one or two architecture dedicated iterations as part of the project's risk reduction.)

Governance or Governess?

The second aspect of deploying an architecture is about ensuring governance – making sure the architecture is followed. As noted in the opening, governance raises several interesting issues:

- How do we know someone does not adhere to the architecture?
- What do we do once we find such a violations?
- Can we avoid architecture violations in the first place? And if yes – how?
- How involved should the architect be?

Before looking into what can be done, let's take another look at the root cause of the problem -- why would anyone want to circumvent, change, or altogether ignore the (hopefully) wonderful architecture you created, even if you kept it relevant (following the advice above J)

Some people would do that because they have other interests which conflict with yours; Others may just don't care -- most people, however, are concerned with the project success just as much as you are but:

- They flipped the bozo on you -- they think they know better.
- They don't see the big picture and they try to optimize locally.
- They don't know any better, they just do what they always did.
- They believe they are following your design but they didn't really understand what you've meant (or alternatively -- you didn't explain yourself very well).
- They cut corners to meet deadlines (just to appease the project manager).
- And there are probably many other “excuses”

The first thing to do, nay, the prerequisite, for understanding there is a problem is to stay involved with the project. Designing an architecture (as elegant and great as it may be) and disappearing will most likely (if not always) will not cut it. As time goes by, and the project progresses, the architecture will deteriorate. How involved should you be? There is no magic number I know of, in my experience the first third of the project as well as the iteration(s) when approaching a release milestone are more prone to trouble vs. the rest of the project but your millage may vary.

Things you can do to find existing violations include reviews (code and design), pair programming (with the developer in the driving seat most of the time and the architect sitting with her), staying approachable (make people feel at ease talking to you), positioning yourself as a "mentor" or

"coach" (someone people would want to come to when they don't understand or know something). You can see I only mentioned one command-and-control approach vs. three collaborative approaches -- even though the task is to “enforce the architecture” or “achieve governance” -- getting there by collaboration (even though it takes more effort on your part) is always better. Even if you do get your way by pulling rank or whatever, people will not like working with you in the long run. (At least that was my experience until, I changed my ways.)

When you find that something is amiss, you sometimes need to be curt, right the wrong, and be done with it. As a general rule, however, try to avoid that. Instead try explaining your decisions, listen to other views, and try to persuade you are right. Developers are smart people. They usually can understand the bigger picture (they just don't have the time to attend to that). Again, this is more effort on your side, but it pays off in the long run.

I don't think you can avoid all "violations", but to minimize violations you need to be aware that they can happen, be attentive for what happens in your project and try to work on your soft-skills, like leadership, organizational politics, communications and human relations. (I'll cover both communications and human relations soon on my DDJ blog)

Summary

The key takeaway here is that an architect cannot just dump an architecture on an unsuspecting development team and disappear. You need to stay involved in the project after the moment in time where you declare the architecture "ready". This deployment of the architecture may not take 100% of your time (though in a large, multi-teamed project it might take more than that J). If you still have "free" time, you can use that to participate in the coding or work with additional projects -- anything goes -- as long as you don't neglect your responsibilities as an architect for the project and help it meet its quality goals.