Documenting Architecture Decisions

Michael Nygard - November 15, 2011

Context

Architecture for agile projects has to be described and defined differently. Not all decisions will be made at once, nor will all of them be done when the project begins.

Agile methods are not opposed to documentation, only to valueless documentation. Documents that assist the team itself can have value, but only if they are kept up to date. Large documents are never kept up to date. Small, modular documents have at least a chance at being updated.

Nobody ever reads large documents, either. Most developers have been on at least one project where the specification document was larger (in bytes) than the total source code size. Those documents are too large to open, read, or update. Bite sized pieces are easier for for all stakeholders to consume.

One of the hardest things to track during the life of a project is the motivation behind certain decisions. A new person coming on to a project may be perplexed, baffled, delighted, or infuriated by some past decision. Without understanding the rationale or consequences, this person has only two choices:

1. **Blindly accept the decision.**
   This response may be OK, if the decision is still valid. It may not be good, however, if the context has changed and the decision should really be revisited. If the project accumulates too many decisions accepted without understanding, then the development team

becomes afraid to change anything and the project collapses under its own weight.
2. **Blindly change it.**
   Again, this may be OK if the decision needs to be reversed. On the other hand, changing the decision without understanding its motivation or consequences could mean damaging the project's overall value without realizing it. (E.g., the decision supported a non-functional requirement that hasn't been tested yet.)

It's better to avoid either blind acceptance or blind reversal.

Decision

We will keep a collection of records for "architecturally significant" decisions: those that affect the structure, non-functional characteristics, dependencies, interfaces, or construction techniques.

An architecture decision record is a short text file in a format similar to an Alexandrian pattern. (Though the decisions themselves are not necessarily patterns, they share the characteristic balancing of forces.) Each record describes a set of forces and a single decision in response to those forces. Note that the decision is the central piece here, so specific forces may appear in multiple ADRs.

We will keep ADRs in the project repository under doc/arch/adr-NNN.md

We should use a lightweight text formatting language like Markdown or Textile.

ADRs will be numbered sequentially and monotonically. Numbers will not be reused.

If a decision is reversed, we will keep the old one around, but mark it as superseded. (It's still relevant to know that it *was* the decision, but is *no longer* the decision.)

We will use a format with just a few parts, so each document is easy to digest. The format has just a few parts.

Title These documents have names that are short noun phrases. For example, "ADR 1: Deployment on Ruby on Rails 3.0.10" or "ADR 9: LDAP for Multitenant Integration"

Context This section describes the forces at play, including technological, political, social, and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.

Decision This section describes our response to these forces. It is stated in full sentences, with active voice. "We will …"

Status A decision may be "proposed" if the project stakeholders haven't agreed with it yet, or "accepted" once it is agreed. If a later ADR changes or reverses a decision, it may be marked as "deprecated" or "superseded" with a reference to its replacement.

Consequences This section describes the resulting context, after applying the decision. All consequences should be listed here, not just the "positive" ones. A particular decision may have positive, negative, and neutral consequences, but all of them affect the team and project in the future.

The whole document should be one or two pages long. We will write each ADR as if it is a conversation with a future developer. This requires good writing style, with full sentences organized into paragraphs. Bullets are acceptable only for visual style, not as an excuse for writing sentence fragments. (Bullets kill people, even PowerPoint bullets.)

Status

Accepted.

Consequences

One ADR describes one significant decision for a specific project. It should be something that has an effect on how the rest of the project will run.

The consequences of one ADR are very likely to become the context for subsequent ADRs. This is also similar to Alexander's idea of a pattern language: the large-scale responses create spaces for the smaller scale to fit into.

Developers and project stakeholders can see the ADRs, even as the team composition changes over time.

The motivation behind previous decisions is visible for everyone, present and future. Nobody is left scratching their heads to understand, "What were they thinking?" and the time to change old decisions will be clear from changes in the project's context.

Experience Report

You may have noticed that this post is formatted like an ADR itself. We've been using this format on a few of our projects since early August. That's not a very long time in the global sense, but early feedback from both clients and developers has been quite positive. In that time, we've had six to ten developers rotate through projects using ADRs. All of them have stated that they appreciate the degree of context they received by reading them.

ADRs have been especially useful for capturing longer-term intentions. We have several clients who are stabilizing their current systems, but looking toward a larger rearchitecture in the not-too-distant future. By writing these intentions down, we don't inadvertently make those future changes harder.

One potential objection is that keeping these in version control with the code makes them less accessible for project managers, client stakeholders, and others who don't live in version control like the development team does. In practice, our projects almost all live in GitHub private repositories, so we can exchange links to the latest version in master. Since GitHub does markdown processing automatically, it looks just as friendly as any wiki page would.

So far, ADRs are proving to be a useful tool, so we'll keep using them.

More Reading

Thanks to Philipe Kruchten for discussing the importance of architecture decisions. I'm told there is more about them in Documenting Software Architectures which is near the top of my reading queue.