Improving your Architecture Documentation (Part One): Two Basic Rules

6 September 2013 | **Business Innovation, Software Engineering** | **Mathias Weyel**
Reading time: 4 minutes

The project is over. For months, you and your team have been working and you are proud of your work. The features are implemented and the list of open bugs is short. You have just applied the last update to the architecture documentation and everything is ready to be taken over by another team to continue working on it.

Half a year later, the phone rings. The new development team shows a steady decline in performance, bugs pile up although more and more effort is spent on hunting them and the team increasingly blames the „horribly convoluted“ architecture for their difficulties. You are to jump to the rescue.

Curious to what might have gone wrong you fire up your computer and take a look at the code. It takes only minutes to see what has happened. All over the place, existing patterns haven’t been followed properly and dozens of references and dependencies have been created between components that had been meticulously decoupled before. No wonder they’re running into problems all the time.

Sounds familiar? Well, so what went wrong? And, far more importantly, what can we do to prevent this from happening?

Let’s first have a look at what we actually want to achieve. When handing over a software system for further development and maintenance, we want to make sure that the architecture is kept stable. This means that new components should be structurally built like the ones before and they have to adhere to the rules the architecture dictates for the interactions between the different components.

Architecture Documentation: Guide your readers
What an architecture documentation actually does, is to describe the architecture of the system by giving an overview over the main components of the system and their relationships between one another. The point where this falls short is where the reader is looking for an answer on how to do things. Diagrams with boxes and arrows give a rough guidance at best. Referring to the existing code not only bears some risk of using just this one component where (for whatever reason) a special solution had been chosen as template but it may also be hard to decide, which parts of the code are implementations of architectural patterns and which ones are specific to the component at hand.

Therefore, instead of just describing the things that are already in place, a good architecture documentation should also provide information on how to add or change things.

This can be done by showing the locations where new components and extensions are expected and how they should be integrated. One or more code snippets with comments should do the trick just fine.

Provide clear rules
However, this is not the full story. As development on an application goes on, inevitably the point will be reached where things have to be added at places where this had not been foreseen. The architecture has to evolve as well. Even this case can somewhat be prepared for in order to avoid ensuing chaos.

Assuming your architecture is based on a handful of basic principles that by all means should be followed, give a list of rules that describe these principles in a “Always do this” and “Never do that” fashion.

Do repeat yourself
At this point, don’t be afraid of two things: redundancy and self-evidence. Your diagrams may already implicitly state a thing or another by having an arrow here or not having an arrow there but if you can put it in form of a rule, it will be much harder to miss. Likewise, there will be things you think should go without saying. State them anyway. To your reader who is not as familiar with the system as you, those things might come a lot less natural than expected.

Conclusion

So there you have it: Two simple rules on how to improve your architecture documentation:

1. Give examples on how to add changes to existing code at the intended places.

2. Give a rule set on how to comply with the architecture on a broader scale to enable unforeseen additions without breaking the architecture principles.

What are your ideas on how to improve architecture documentation to be more valuable for those taking over projects?