About Software Architectures and the IEEE 1471 Standard

Adrian Kosmaczewski

2005-01-16

Looking for information on the topic of Software Architecture, I came accross the IEEE 1471 Standard, the "IEEE Recommended Practice for Architectural Description of a Software-Intensive System". I must admit that I had never heard of it before. And I think there is a reason for this.

The home page of the standard, at http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html does not really give good hints about what is the standard about, but a quick search in Google has given me enough information to post here a quick summary of it.

As far as I understood, the IEEE 1471 is a set of rules showing a common framework of methodologies and vocabulary useful to describe any "Software-Intensive" system. The expression "Software-Intensive" carries the idea that not every software project needs a formal description of its architecture (although I firmly believe that there is an implicit, hidden architect buried inside every software developer).

Before continuing with the topic of the IEEE 1471 standard, I would like to point out some facts that I feel important about Software Architecture.

For larger systems, it is fundamental to have a formal document that describes the architecture of the software being developed; and by "larger" I mean any software that is being developed by more than one person. Simply put, Software Architecture becomes self-evident when you develop version 2.0 of any software: if you can add the new features of the system easily, without too much trouble, integrating them with the old code base and even enhancing the old features, then your architecture simply rocks. Otherwise, it was screwed from the beginning.

From the previous paragraph you can infere that Software Architecture is tightly related to the evolution of software systems; how many times you have developed something to find out that you just cannot expand the initial solution a couple of months later after the initial release? That is where you start wondering about Design Patterns, Software Architecture and plenty of other buzzwords.

But I think that this goes much further: Software Architecture is a way to describe a dynamic, comprehensive, secure and manageable system; I will describe each of these keywords and what are the implications.

1. By "Dynamic" I do not mean that your algorithms adapt well or that your exception handling scheme has great elegance (which is fine): when I say dynamic, I mean that you can adapt to the needs of your customer. Sooner or later, your customer will come up to you asking for new features. And you should (if your analysis shows that they are feasible) integrate them in the software with the least possible cost, both in terms of money and time. This is the kind of dynamism that Software Architectures enable.

2. By "Comprehensive", I mean that the solution proposed by the Architecture of any system should, at its very least, contain a reasonable solution for every problem raised by the creation of the software. Software development is a pretty hard task, which not only can solve a problem, but can create many as well (deployment, security, deadlines, bugs, interoperability, change requests, etc). A Software Architecture has at least one proposed solution for any of these; fortunately, as time goes by, we learn more and more from the mistakes of the past, and this enables us to create better solutions in every aspect.

3. By "Secure", I do **not** mean paranoia or marketing chit chat à la Microsoft; I mean *conscience*. Software integrators usually just have to know what are the risks and the threats to which the software used to create solutions is exposed: bugs in application and web servers, security holes in operating systems (no no, I am not thinking of any operating system in particular) and issues like that. But ISVs (Independent Software Vendors) that provide innovative solutions, used by software integrators, have to take special care and actively maintain a security policy that enables prompt solutions to any possible threat. This is something that is poisoning Microsoft lately.

4. Finally, by "Manageable" I mean all the tasks that make your development team keep smiling throughout the project: coding standards, clear interfaces, whiteboards with colorful diagrams, well-written and updated documentation, a one-step build/deployment procedure, a clear vision of the product lifecycle, and free soda or coffee machines. This will enable you, as a Software Architect, to keep the project up and running until the last release, even if your team changes, even if you leave the company before the project is finished.

Every one of the four aspects I have defined above are good subjects for articles of their own... watch out for improvements in the future.

Getting back to the IEEE 1471 standard: an important fact of it is that it is a "recommended practice": in fact, it just proposes a terminology and a set of concepts to describe an architecture, but it does not even say how to do this description in actual facts. It does not require nor define a particular description language, it does not offer any completeness criteria, and it does not force a minimal set of information in the description of the Architecture.

But it does give a formal definition for the word “Architecture”:

> *“The fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.”*

Wikipedia also has something to say about this. And Bredemeyer Consulting has an impressive set of documentation.

The following graph shows the conceptual parts of the IEEE 1471 standard:

(Diagram taken from the paper at http://www.ifi.uio.no/~mmo/generic/papers/IEEE.pdf)

The central concept in the standard is the separation between the concepts of “Architecture” and “Architecture Description”; this is, I think, the most important contribution of the standard, the latter being the concrete representation of the first as a document or a set of documents.

The concepts of “Stakeholders” (not “S**k**a**t**eholder” which is actually something different) is fundamental as well: these are the roles or groups of people that might have any intervention on the creation, usage and/or management of the system, at any given level (this spans from the Developers to the Customers and the Final Users, but also embodies the Software Architect himself!). This “interest on the system” is described in the above diagram as “Concerns”, and they define a particular “Viewpoint” on the system itself.

The Architecture Description is described as a series of “Views”, which are closely related to the Stakeholders: each one of them holds a particular view on the system, and both concepts are thus closely related. For the sake of top-down decomposition of the problem domain, each View can be subdivided into several “Models”.

Finally, each Stakeholder has “Viewpoints” over a certain View: the Architecture Description identifies the Stakeholders, their concerns, and offers one and exactly one View for each Viewpoint.

Then the problem is to define this (rather fuzzy) line between the View and its Viewpoint: that is why the IEEE 1471 offers several guidelines used to define Viewpoints and their context, how to organize them and how to explain any inconsistencies among them.

For an example of a practical use of the standard in the software industry, this paper (44 KB) offers an excellent insight (taken from http://www.mrtc.mdh.se/php/publ_show.php3?id=0529)

Sofware Architecture is not a new topic: Dijkstra has even talked about it in the sixties. But the formalization of its description could enable a better communication between software developers and architects, with the well-known risk that any standardization brings a reduction in creativity; I think that there is a risk, but that the benefits are interesting anyway.