Sidebar – Software Architectures

Patricia Lago and Hans van Vliet

Vrije Universiteit, Amsterdam, The Netherlands {patricia, hans}@cs.vu.nl

Software architecture is becoming one of the central topics in software engineering. In early publications, such as [Sha88], software architecture was by and large synonymous with global design. In [SG96] we read "the architecture of a software system defines that system in terms of computational components and interactions among those components". In a traditional software engineering process, during the software design phase the system is decomposed into a number of interacting components (or modules). The top-level decomposition of a system into major components together with a characterization of how these components interact, was considered as the software architecture of the system under development. In this respect requirements engineering is an activity focussing very much on the problem space, while the subsequent design phase focuses on the solution space. We call this a *pre-architecture development approach*. Here, few persons (also called stakeholders) are typically involved (like for example the project manager, robotic expert, the software analyst, few developers). Iteration involves functional requirements only: once agreed upon, these are supplemented with non-functional requirements to form the requirements specification used as input for design. In particular, there is no balancing between functional and non-functional requirements

Conversely, modern software engineering is moving to an *architecture-centric development approach*. Here, the discussions involve multiple stakeholders: the project manager, different classes of robotic and software experts, future maintainers of the system, owners of other inter-operating systems and services. Iteration concerns both functional and non-functional requirements. In particular, architecting involves finding a balance between these type of requirements. Only when this balance is reached, next steps can be taken.

In the latter approach, software architecture has to bridge the gap between the world of a variety of, often non-software-technical stakeholders on one hand – the problem space –, and the technical world of software developers and designers on the other hand – the solution space.

Software architecture hence describes much more than just the components and the interactions among them. It serves three main purposes [BCK03]:

D. Brugali (Ed.): Software Eng. for Experimental Robotics. STAR 30, pp. 253–256, 2007.

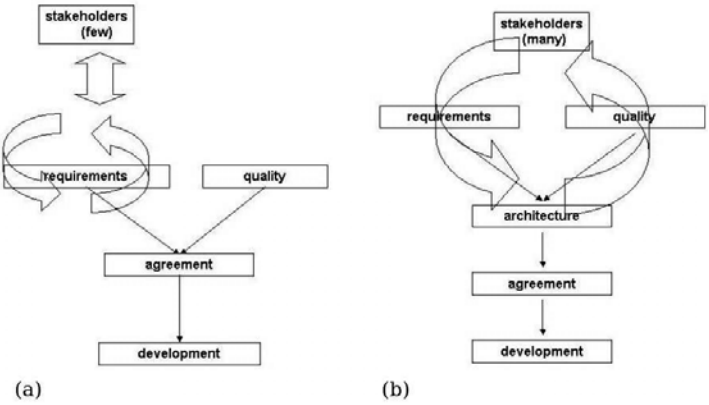

Fig. 1. The software life cycle.

- It is a vehicle for communication among stakeholders. A software architecture is a global, often graphic, description that can be communicated with the customers, end users, designers, and so on. By developing scenarios of anticipated use, relevant quality aspects can be analyzed and discussed with various stakeholders.
- It captures early design decisions. In a software architecture, the global structure of the system has been decided upon, through the explicit assignment of functionality to components of the architecture. These early design decisions are important since their ramifications are felt in all subsequent phases. It is therefore paramount to assess their quality at the earliest possible moment. By evaluating the architecture, a first and global insight into important quality aspects can be obtained. The global structure decided upon at this stage also structures development: the work-breakdown structure may be based on the decomposition chosen at this stage, testing may be organized around this same decomposition, and so on.
- It is a transferable abstraction of a system. The architecture is a basis for reuse. Design decisions are often ordered, from essential to nice features. The essential decisions are captured in the architecture, while the nice features can be decided upon at a later stage. The software architecture thus provides a basis for a family of similar systems, a so-called *product line*. The global description captured in the architecture may also serve as a basis for training, e.g. to introduce new team members.

We do not want here to give yet another definition of what software architecture is or should be. For the interested reader [SEI06] provides a quite complete overview of what is meant by software architecture around the world.

Recently, we observed an interesting evolution of this widely discussed concept. In a more traditional connotation Software Architecture (SA) is con-

sidered as the pure, technical result of a software engineering activity: in this sense, it is limited to the modeling of the technical and technological aspects of a computerized information system. SA however is broader than that. Modeling the 'technical SA' is of course needed [Med05]. But to make sense, this 'technical SA' must be integrated with its environment, and this includes non-technical aspects too, like organisation, geographical location, economical, legal aspects, etc. Some efforts have been done in the literature to illustrate these aspects. Architectural standards [IEE00] and reference frameworks [Kru04a, HNS99, Zac87, Oa00] all propose some views showing e.g. business or process aspects next to more technical ones. Still, they all remain focused at the technical level. It remains difficult if not impossible to capture technical and non-technical aspects in an *integrated* way [Kru04b, TA05]. Interesting research questions are, for instance: How does a change in the market influence the architecture of an organization? If the company closes an agreement with a different technology vendor, which parts of the technical architecture are potentially influenced? How much will this cost me? To answer these questions, SA should evolve to cover all aspects in an integrated way. Integration of various technical aspects has been an issue, too. For example, the relation between SA and quality requirements has been investigated for a long time. Relevant results in this respect are represented by the use of tactics in the Architecture Tradeoff Analysis method [BCK03]. This allows for example to analyze how performance is influenced by certain architecture decisions.

A further issue is to keep SA alive. We believe that SA should capture what is likely not to change, or better what is difficult to change. This means that SA gives the relatively long-lasting picture of a software system. Nonetheless, when changes occur (and they will! [LvV05, Fow03]) we must be able to keep the architecture aligned. This does not happen by itself. In practice, organizations have difficulties to align software architecture with the *real situation*. There are various reasons for that. Architects do not work together with those making the changes, hence remain isolated in their ivory tower; sometimes the way SA is documented is not understood, or it differs from the way the various actors are used to document the results of their production processes. Updating SA would require the allocation of additional effort, and such effort is expensive and does not bring an immediate advantage. Hence it is perceived as a loss of time.

References

[BCK03] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, SEI Series in Software Engineering, Addison-Wesley, second edition, 2003.

[Fow03] M. Fowler, *Who needs an architect?*, IEEE Software **5** (2003), no. 20, 11–13.

[HNS99] C. Hofmeister, R. Nord, and D. Soni, *Applied software architecture*, Addison-Wesley, 1999.

[IEE00] IEEE, *Ieee recommended practice for architectural description of software-intensive systems*, IEEE Standard 1471-2000, 2000.

[Kru04a] P. Kruchten, *Architectural blueprints - the 4+1 view model of software architecture*, IEEE Software **21** (2004), no. 6, 42–50.

[Kru04b] P. Kruchten, *An ontology of architectural design decisions in software-intensive systems*, Intern. Workshop on Software Variability Management (2004).

[LvV05] P. Lago and H. van Vliet, *Explicit assumptions enrich architectural models*, IEEE 27th International Conference on Software Engineering (ICSE) (2005), 206–214.

[Med05] N. Medvidovic, *Software architectures and embedded systems: a match made in heaven?*, IEEE Software **22** (2005), no. 5.

[Oa00] H. Obbink and al., *Copa: a component-oriented platform architecting method for families of software-intensive electronic products*, Tech. report, http://www.extra.research.philips.com/SAE/COPA/, 2000.

[SEI06] CMU SEI, *How do you define software architecture?*, Tech. report, http://www.sei.cmu.edu/architecture/definitions.html, 2006.

[SG96] M. Shaw and D. Garlan, *Software architecture, perspectives on an emerging discipline*, Prentice-Hall, 1996.

[Sha88] M. Shaw, *Toward higher level abstractions for software systems*, Proc. Tercer Simposio Internacional del Conocimiento y su Ingegneria (1988).

[TA05] J. Tyree and A. Akerman, *Architecture decisions: Demystifying architecture*, IEEE Software **22** (2005), no. 2, 19–27.

[Zac87] J.A. Zachman, *A framework for information systems architecture*, IBM SYSTEMS JOURNAL **26** (1987), no. 3.