Engineering Evolution for software's future: an architectural approach

Dalila Tamzalit

Engineering Evolution for software's future: an architectural approach.

Dalila Tamzalit

University of Nantes, LINA, CNRS UMR 6241

Nantes, France.

Abstract

Software evolution is often an activity conducted on a case by case basis, leaning generally on feelings, by trial and error, intuition and experience. However, this approach is nowadays known as time consuming, risky, error-prone and very costly. Significant efforts have been done since Lehman and his famous laws, but abstracting software evolution for an efficient evolution management is still in its infancy. In fact, even if more controlled, software evolution is still an ad hoc activity and generally took into consideration only when an evolution need occurs. It is rarely anticipated nor reused. It is now time to go two steps beyond: *(i) engineering software evolution* and *(ii) offering evolution capabilities* not only for current systems but also for legacy and future systems. We propose, through some prospective ideas, to support the first objective by abstracting and manipulating evolution as a first-class entity and to support the second one by leaning on an architectural approach, outlining thus some important research tracks.

Engineering Evolution for software's future: an architectural approach.

Engineering Software Evolution: evolution as a first-class entity

As it was already pointed out by the NATO conference in 1968 and formalized by M.M. Lehman (?, ?), two main reasons of producing more and more large software are the fact that software have an intrinsic increasing complexity and that they are no more than living entities continuously evolving in an infinite life cycle. In response to changing needs (from stakeholders, the external environment, the introduction of new technologies ...), the software engineer needs to manage and evolve the software system as much as possible according to the overall system's goals, rather than modifying isolated parts of the system in an *ad hoc* way. According to this growing complexity, it is crucial to handle evolution needs in a rigorous and disciplined manner. Software evolution needs to be analysed, specified, guided, controled, capitalised and also reused and made reusable. It needs to be *abstracted* and *engineered* like any other software artifact: evolution is thus seen as a *first-class entity*.

In light of these facts, software evolution needs to be included in the software's development cycle. It should be considered as the other requirements, analysed, specified, designed, validated and ultimately implemented and deployed. Like any engineered software artifact, software evolution needs to be surrounded with theoretical and practical approaches by means of (meta) modelling, guidelines and methodologies. This gives an evolution dimension to a software. Each system can thus be enriched with an evolution dimension.

Supporting evolution during the software life cyle: an architectural approach.

The most stable and essential part of a software system is its strategic objectives and not the way they are realised. The realisation of a software system is generally less stable than the essential part. In order to ensure system's longevity, this essential part needs and should be decoupled from its realisation by explicitly specifying both of them and their existing interrelations. Software architectures are an ideal mean to specify the heart of a system and to decouple it from its different facets thanks to their powerful abstraction mechanism (Perry & Wolf, 1992 ; Bass, Clements, & Kazman, 1997 ; Shaw & Garlan, 1996). In fact, they allow, according to the conceptual framework of the IEEE 1471 Standard(Hilliard, 2000), to express all facets of a software system and are generally technological-free. It represents the pivot of a system as well for their design as for their evolution. Each facet can be handled in its own evolution but also all dependent and related facets can be consequently evolved if needed, following their explicit interrelations expressed within the architecture. This is an important feature since large systems must continuously evolve to respond to external and internal pressures.

Even if evolution needs concern legacy systems, current eroded systems or future systems, the evolution can be handled in a uniform way since the pivot is their software architecture. Thereby, each software system, whatever its age, can benefit from disciplined evolution through its architecture: the software architecture of a legacy-system can be obtained by way of recovering, the one of an erroded system can be retro-engineered and a future system will be designed first in its architecture. Managing evolution by engineering it and by leaning on architectural descriptions represent a force to handle the evolution of any kind of system in a uniform way.

General discussion

Two major objectives have been prospected in this paper: *(i) engineering software evolution* and *(ii) offering evolution capabilities for old, current and future software systems*. Even if awareness about abstracting software evolution is already underway (Jazayeri, 2005 ; Le Goaer, Tamzalit, Oussalah, & Seriai, 2008 ; Garlan, 2008 ; Tamzalit & Mens, 2010), this trend should be strengthened to strive towards *engineering software evolution*. Evolution must be in mind of architects and designers from the beginning. They must think about the system's requirements and include evolution requirements of the system to develop. These requirements must also be analysed. Of course, foreseen evolution are easier to handle than unforeseen ones. For this reason, the system should be thought with evolution capabilities and not only by storing some known-evolution scenarios. Analysis of evolution should consider these both situations. Meta-models, models, mechanisms, laws and guidelines are the key points to specify and handle properly any evolution. In addition, since evolution is a first-class entity, it can be stored as *evolution patterns* and reused in similar situations (Tamzalit & Mens, 2010).

Thanks to engineering software evolution and thanks to considering the software architecture as the evolution pivot, it is possible to extend significantly system's life expectancy. This allows old systems to evolve for current needs, existing systems to evolve and prepare future evolutions and future systems. In fact, we are convinced that engineering software evolution only in the present has no sense if the past cannot come into the present and if the future cannot be considered in the present. Software's future needs to lean on software's past and present.

Références

Bass, L., Clements, P., & Kazman, R. (1997). *Software architecture in practice.* Addison-Wesley Professional.

Garlan, D. (2008). Evolution Styles Formal Foundations and Tool Support for Software Architecture Evolution. *Tech. report. CMU-CS-08-142, CMU*.

Hilliard, R. (2000). *ANSI/IEEE Stand. 1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems* (Rapport technique). IEEE. Disponible sur `http://standards.ieee.org`

Jazayeri, M. (2005). Species evolve, individuals age. In *Principles of software evolution, eighth international workshop on* (pp. 3–9).

Le Goaer, O., Tamzalit, D., Oussalah, M., & Seriai, A. (2008). Evolution Shelf: Reusing Evolution Expertise within Component-Based Software Architectures. *IEEE COMPSAC*, 311–318.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Softw. Eng. Notes*, *17*(4), 40–52.

Shaw, M., & Garlan, D. (1996). Software architecture — perspectives on an emerging discipline.

Tamzalit, D., & Mens, T. (2010). Guiding architectural restructuring through architectural styles. *IEEE ECBS 2010: 17th IEEE International Conference on Engineering of Computer-Based Systems*, 69–78.