Extended Abstract
Software Architecture Evolution – An Integrated Approach in Industry

Hongyu Pei Breivold*, Ivica Crnkovic**
*ABB Corporate Research, Industrial Software Systems, Västerås, Sweden.
**Mälardalen University, Västerås, Sweden.
Emails: hongyu.pei-breivold@se.abb.com, ivica.crnkovic@mdh.se

Abstract

To improve the capability in being able to understand and analyze systematically software architecture evolution, we introduced in our earlier work a software evolvability model and software architecture evolvability analysis method. This extended abstract reports the integration of the evolvability model and evolvability analysis method in an industrial context.

Keywords: Software evolvability, Software evolution, Software architecture.

1 Introduction

Software evolution is characterized by inevitable changes of software and increasing software complexities, which in turn may lead to huge costs unless rigorously taking into account change accommodations. Software evolvability has thus been recognized as a fundamental element for increasing strategic and economic value of software, as it "*bears on the ability of a system to accommodate changes in its requirements throughout the system's lifespan with the least possible cost while maintaining architectural integrity*" [4]. This is in particular true for long-lived systems. For such systems, there is a need to address evolvability explicitly, carry out software evolution efficiently, and prolong the productive life of the software systems. As software architecture holds a key to the possibility to implement changes in an efficient manner [1], software architecture evolution has become an integral part of software lifecycle.

2 An Integrated Approach

To improve the capability in being able to understand and analyze systematically software architecture evolution, we introduced a software evolvability model [2], in which subcharacteristics of software evolvability and corresponding measuring attributes are identified. The subcharacteristics that are of primary importance for software evolvability in a given context (long-lived software-intensive systems) are: analyzability, architectural integrity, changeability, extensibility, portability, testability and domain-specific attributes. In addition, we also introduced a structured method for analyzing evolvability at the architectural level, i.e. the ARchitecture Evolvability Analysis (AREA) method [3].

The evolvability model is a way to articulate subcharacteristics for an evolvable system that an architecture must support. The evolvability analysis method starts with identification of change stimuli and guides architects through the analysis of potential architectural requirements that the software architecture needs to adapt to, as well as their implications. Integration of the evolvability model and the analysis method ensures that the implications of the potential requirements, improvement strategies and evolution paths of the software architecture are assessed systematically with respect to evolvability subcharacteristics. Fig. 1 illustrates the process flow of the integrated approach, showing the activities (in the squares) and corresponding input/output artefacts.

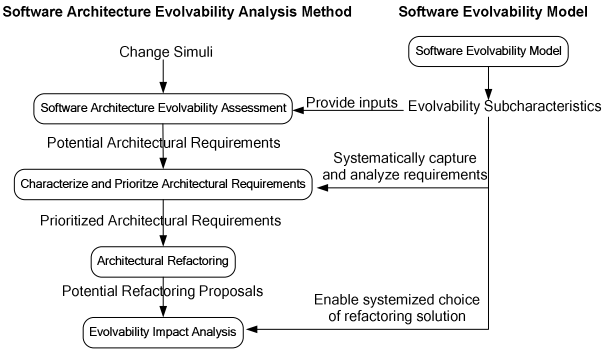

Fig. 1 Process workflow of the integrated approach

3 Managing Software Architecture Evolution at ABB

The integrated approach that embodies both the software evolvability model and analysis method was applied to assess an industrial automation control system. A change stimulus from the business perspective was the need for distributed application development.

3.1 Background

The automation control system consists of more than three million lines of C/C++ code. All the source code was compiled into a single binary software package, which consists of various software applications, aiming for specific tasks that enable the automation controller to handle various applications. The main problem with the software architecture was its monolithic characteristic with the existence of tight coupling between some components that resided in different layers. As a consequence, source code updates had to be done not only on the application level, but through several layers, several subsystems and components. As the system was expanding, it became more difficult to

ensure that the modifications of specific application software would not affect the quality of other applications. This constituted a bottleneck in the effort to enable distributed application development.

3.2 Activities

The refactoring process was performed through close collaboration between the corporate research team and local software development organization. It was a continuous maturation process that took approximately one calendar year including analysis, identification of architecture evolution path and refactoring of some primary components. The identification and analysis of potential architectural requirements was performed by architecture core team which consists of 6-7 persons. These architectural requirements were categorized and checked according to the subcharacteristics identified in the software evolvability model. This was to justify whether the identified requirements have covered primary evolvability aspects, and whether the realization of each requirement would lead to an improvement of the subcharacteristics or possibly a decrease, which would then require a tradeoff decision.

Subsequently, 2-3 persons in architecture core team identified refactoring solution proposals that were discussed and documented in terms of the following views: (i) problem of the original design of the component; (ii) new requirements that the component needs to fulfill; (iii) architectural solution to design problems; (iv) rationale of the solution proposal and architectural implications of the deployment of the component on quality attributes; and (v) estimated workload for implementation and verification. The architectural implications were assessed with respect to the subcharacteristics identified in the software evolvability model as well. These proposals were discussed with the main technical responsible persons and architects at the development organization, documented as evolution path for the architecture, circulated among the core team for inspection, and transferred further to the implementation teams.

3.3 Results

By integrating the evolvability analysis method and the evolvability model, potential architectural requirements for future changes were captured and analyzed in a systematic way. Likewise, by analyzing improvement proposals with respect to their implications on evolvability subcharacteristics, we further avoided an ad hoc choice of potential evolution paths of software architecture. The refactoring proposal report highlighted major software components that exhibited architectural shortcomings. Positive experience was that the architecture requirements, corresponding architectural decisions, rationale and architecture evolution path became more explicit, better founded and documented. The refactoring improvement proposals were widely accepted by the involved stakeholders. The defects and refactoring proposals were reported to the development organization which started corresponding improvement activities by the implementation teams.

3.4 Experiences and Lessons Learned

Throughout the architecture refactoring process, reasonable and measurable targets were set up to constantly monitor the progress. For instance, a metric was the number of exposed public interfaces. Monitoring of this metric was conducted on a regular interval. It provided signal indication on analyzing the reason for trend of increasing number of interfaces when this happened. This in turn provided a source of input progress measurement and risk judgments.

Incremental architecture transformation strategy was a preferred choice with the intention of not to disrupt the ongoing development projects. The criteria for prioritization of potential architectural requirements were thus set up as: (i) enable building of existing types of extensions after refactoring and architecture restructuring; and (ii) enable new extensions and simplify interfaces that may have negative effects on implementing new extensions. Accordingly, components that needed to be refactored could be categorized into different priorities. A sequence of incremental code transformation steps was identified, performed and verified.

One aspect that may be further improved in the integrated approach is that the determination of potential refactoring proposals is on a qualitative level in terms of their impact on evolvability subcharacteristics. In addition, the importance of evolvability subcharacteristics has been treated implicitly, i.e. the choice of prioritized architectural requirements qualitatively and implicitly set weight on these subcharacteristics. Explicit quantitative assessment remains to be done in the future work.

4 Conclusions and Future Focus

Following the described experiences, we will continue with the integrated approach for software evolvability assessment in other domains. We also plan to extend the evolvability analysis method with weight score computation to quantitatively define relative preferences on evolvability subcharacteristics provided by different stakeholders, and quantitatively weight how refactoring alternatives support these subcharacteristics.

References

[1] Bass, L., Clements, P., and Kazman, R.: ‘Software Architecture in Practice’, Addison-Wesley Professional, 2003.

[2] Breivold, H.P., Crnkovic, I., and Eriksson, P.J.: ‘Analyzing Software Evolvability’, COMPSAC 2008.

[3] Breivold, H.P., Crnkovic, I., Land, R., and Larsson, M.: ‘Analyzing Software Evolvability of an Industrial Automation Control System: A Case Study’, ICSEA 2008, pp. 205-213.

[4] Rowe, D., Leaney, J., and Lowe, D.: ‘Defining systems evolvability- a taxonomy of change’, ECBS 1998.