Analyzing Software Evolvability

Hongyu Pei Breivold[1,2], Ivica Crnkovic[2], Peter J Eriksson[1]
[1]*ABB AB, Corporate Research*
{hongyu.pei-breivold, peter.j.eriksson}@se.abb.com
[2]*Mälardalen University*
{ivica.crnkovic, hongyu.pei-breivold}@mdh.se

Abstract

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. This is in particular true for long-lived systems in which changes go beyond maintainability. For such systems, there is a need to address evolvability explicitly during the entire lifecycle. Nevertheless, there is a lack of a model that can be used for analyzing, evaluating and comparing software systems in terms of evolvability. In this paper, we describe the initial establishment of an evolvability model as a framework for analysis of software evolvability. We motivate and exemplify the model through an industrial case study of a software-intensive automation system.

1. Introduction

Software maintenance and evolution are characterised by their huge cost and cumbersome implementation [1]. The systems' capability to cost-effectively accommodate various changes has become essential. Accordingly, there is a strong need to carry out software evolution efficiently and reliably, and prolong the productive life of a software system. In this paper, we use evolution to refer to the particular evolution stage as described in the staged model by Bennett and Rajlich [1]. We refer to the evolvability definition in [18], since it expresses the dynamic behaviour during a software system's lifecycle and supports the staged model: "*An attribute that 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*."

1.1 Motivations

The need to explicitly address software evolvability is becoming recognized [5]. There are examples of different industrial systems that often have a lifetime of 20-30 years. These systems are subject to and may undergo a substantial amount of evolutionary changes, e.g. software technology changes, software systems merge due to organizational changes, demands for distributed development, system migration to product line architecture, etc. The evolution problems we have observed came from different cases. In this paper, we exemplify and analyze in particular one industrial case study that was carried out on a large automation control system at ABB. The controller software consists of more than three million lines of code written in C/C++ and a complex threading model, with support for a variety of different applications and devices. It has grown in size and complexity, as new features and solutions have been added to enhance functionality and to support new hardware, such as devices, I/O boards and production equipment. Such a complex system is difficult to maintain. It is also important and considerably more difficult to evolve. Due to different measures such as organizational and lifecycle process improvements, the system keeps the maintainability, but the evolvability becomes more difficult since the increased complexity in turn leads to decreased flexibility, resulting in problems to add new features. Consequently, it would become costly to adapt to new market demands and penetrate new markets.

Our particular system is delivered as a single monolithic software package, which consists of various software applications developed by distributed development teams. These applications aim for specific tasks in painting, welding, gluing, machine tending and palletizing, etc. In order to keep the integration and delivery process efficient, the initial architectural decision was to keep the deployment artifact monolithic; The complete set of functionality and services is present in every product even though not everything is required in the specific product. As the system grew, it became more difficult to ensure that the modifications of specific application software do not affect the quality of other parts of the software system. As a result, it becomes difficult and time-consuming to modify software artifacts, integrate and test products. To continue exploiting the substantial software investment made and to continuously improve the system for longer productive lifetime, it has become essential to explicitly address evolvability, since the inability to effectively and reliably evolve software systems means loss of business opportunities [1]. We want to emphasize here that the problem raised is not a problem of maintainability. The major problems arise

when brand new (very different) features or different development paradigms, shifting business and organizational goals are introduced, so the problems related to the software evolvability – a fundamental element for increasing strategic and economic value of the software [21].

To solve the problems presented above, we need to handle several research issues: (i) which characteristics are necessary for a software system to be evolvable; (ii) how to assess evolvability in a systematic manner; (iii) how to achieve evolvability; and (iv) how to measure evolvability. Accordingly, we outline a software evolvability model in section 2, where necessary subcharacteristics of software evolvability and corresponding measuring attributes are identified. This model is established as a first step towards analyzing and quantifying evolvability, a base and check points for evolvability evaluation and improvement. Further in section 3, we present the structured way of evolvability evaluation that we used in the case study, and a brief analysis of the evolvability subcharacteristics. Section 4 presents related work. Section 5 concludes the paper and outlines the future work.

2. Software evolvability model

Software evolvability is a multifaceted quality attribute [18]. Based on the definition in [18], the software quality challenges and assessment [8], the types of change stimuli and evolution [4], and experiences we gained through industrial case studies, we have discovered that only having a collection of the subcharacteristics of maintainability as defined in the ISO software quality standard [11] is not sufficient for a software system to be evolvable. Therefore, we have (i) complimented and identified subcharacteristics that are of primary importance for an evolvable software system, and (ii) outlined a software evolvability model that provides a basis for analyzing and evaluating software evolvability. The idea with the evolvability model is to further derive the identified subcharacteristics to the extent when we are able to quantify them and/or make appropriate reasoning about the quality of service, as in Figure 1.

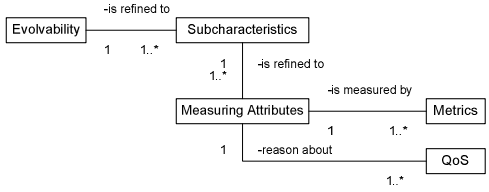

Figure 1 Concept of the evolvability model

The identified subcharacteristics are summarized in Table 1. They are a union of quality characteristics having to do with changes, and are relevant for characterization of evolution of software-intensive systems during their life cycle. With these subcharacteristics in mind, we have a basis on which different systems can be examined and compared in terms of evolvability. Any system that does not explicitly address one or more of these subcharacteristics is missing an element that probably will undermine the system's ability to be evolved.

Table 1 Subcharacteristics of evolvability

Sub-characteristics	Description
Analyzability	The capability of the software system to enable the identification of influenced parts due to change stimuli (based on [11]).
Integrity	The non-occurrence of improper alteration of architectural information (based on [12]).
Changeability	The capability of the software system to enable a specified modification to be implemented and avoid unexpected effects (based on [11]).
Extensibility	The capability of the software system to enable the implementation of extensions to expand or enhance the system with new capabilities and features with minimal impact to the existing system (based on [11]).
Portability	The capability of the software system to be transferred from one environment to another [11].
Testability	The capability of the software system to enable modified software to be validated [11].
Domain-specific attributes	The additional quality subcharacteristics that are required by specific domains [8].

These subcharacteristics serve as a catalog of check points for evaluation. Each subcharacteristic is motivated and explained below in conjunction with the case study. Examples of measuring attributes for each subcharacteristic are given.

Analyzability The release frequency of the controller software is twice a year, with around 40 various new requirements that need to be implemented in each release. These requirements may have impact on different attributes of the system, and the possible impact must be analyzed before the implementation of the requirements. This requires that the software system must have the capability to be analyzed and explored in terms of the impact to the software by introducing a change.

Description: Many perspectives are included in this dimension, e.g. identification and decisions on what to modify, analysis and exploration of emerging technologies from maintenance and evolution perspectives. *Measuring attributes* include modularity, complexity, and documentation.

Integrity A strategy for communicating architectural principles that we found out from various case studies was to appoint members of the core architecture team as technical leaders in the development projects. However, this strategy although helpful to certain extent, did not completely prevent developers from insufficient understanding and/or misunderstanding of the initial architectural decisions, resulting in violation of architectural conformance. This may lead to evolvability degradation in the long run.

Description: Architectural integrity is related to understanding and coherence to the architectural

decisions and adherence to the original architectural styles, patterns or strategies. Taking integrity as one subcharacteristic of evolvability does not mean that the architectural approaches are not allowed to be changed. Proper architectural integrity management is essential for the architecture to allow unanticipated changes in the software without compromising software integrity and to evolve in a controlled way [1]. *Measuring attributes* include architectural documentation.

Changeability Due to the monolithic characteristic of the controller software, modifications in certain parts of the software package may lead to ripple effects, and requires recompiling, reintegrating and retesting of the whole system. This results in inflexibility of patching and customers have to wait for a new release even in case of corrective maintenance and configuration changes. Therefore, it is required that the software system must have the ease and capability to be changed without negative implications or with controlled implications to the other parts of the software system.

Description: Software architecture that is capable of accommodating change must be specifically designed for change [10]. *Measuring attributes* include complexity, coupling, change impact, encapsulation, reuse, modularity.

Portability The current controller software supports VxWorks and Microsoft Windows NT. There is a need of openness for choosing among different operating system vendors, e.g. Linux and Windows CE.

Description: Due to the rapid technical development on hardware and software technologies, portability is one of the key enablers that can provide possibility to choose between different hardware and operating system vendors as well as various versions of frameworks. *Measuring attributes* include mechanisms facilitating adaptation to different environments.

Extensibility The current controller software supports around 20 different applications that are developed by several distributed development centers around the world. To adapt to the increased customer focus on specific applications and to enable establishment of new market segments, the controller, like any other software systems, must constantly raise the service level through supporting more functionality and providing more features [3].

Description: One might argue that extensibility is a subset of changeability. Due to the fact that about 55% of all change requests are new or changed requirements [15], we define extensibility explicitly as one subcharacteristic of evolvability. It is a system design principle where the implementation takes future growth into consideration. *Measuring attributes* include modularity, coupling, encapsulation, change impact.

Testability The controller software exposed huge number of public interfaces which resulted in tremendous time merely on interface tests. One task was therefore to reduce the public interfaces to around 10%. Besides, due to the monolithic characteristic, error corrections in one part of the software requires retesting of the whole system. One issue was therefore to investigate the feasibility of testing only modified parts.

Description: According to statistics [7], software testing spends as much as 50% of development costs and comprises up to 50% of development time. Hence, testability is a key feature permitting high quality to be combined with reduced time-to-market. *Measuring attributes* include complexity, modularity.

Domain- specific attributes The controller software has critical real-time calculation demands. It is also required to reduce base software code size and runtime footprint.

Description: Different domains may require additional quality characteristics that are specific for a software system to be evolvable. *Measuring attributes* depend on the specific domains.

3. Case study

We conducted the following structured evaluation steps shown in Figure 2. The involved stakeholders expressed that they were pleased with this systematic approach, as it made architecture requirements and corresponding design decisions more explicit, better founded and documented.

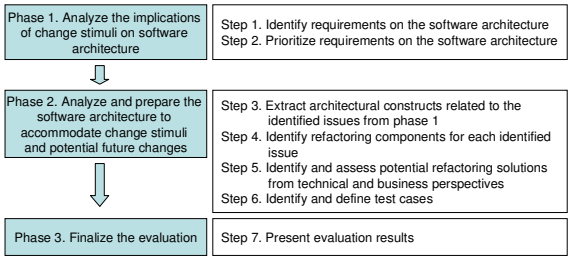

Figure 2 Evaluation steps

The evaluation results included (i) the identified and prioritized requirements on the software architecture; (ii) identified components/modules that need to be refactored for enhancement or adaptation; (iii) refactoring investigation documentation which describes the current situation and solutions to each identified candidate that need to be refactored, including estimated workload; and (iv) test scenarios.

3.1 Analysis of evolvability subcharacteristics

Analyzability was addressed through refining activities for each identified requirement. **Integrity** was addressed through extracting rationale for each design decision; and providing training, guidelines and code examples for software developers and using

tactics that enable the achievement of a certain quality characteristic. **Changeability** was addressed through restructuring the original function-oriented architecture to product-line architecture. **Extensibility** was addressed through the definition of a Base Software SDK (Software Development Kit), consisting of well-documented API (Application Programming Interface), wizards and tools for developing application-specific extensions. **Portability** was handled through the portability layer which encapsulates infrastructure technology choices and provides interfaces for application software in the controller. **Testability** was addressed through defining test scenarios and applications to support platform testing. **Domain-specific attribute** was planned with respect to functionality partition of the controller software.

4. Related work

To evaluate evolvability, Ramil and Lehman proposed metrics based on implementation change logs [16] and computation of metrics using the number of modules in a software system [13]. Another set of metrics is based on software life span and software size [20]. In [19], a framework of process-oriented metrics for software evolvability was proposed to intuitively develop architectural evolvability metrics and to trace the metrics back to the evolvability requirements based on the NFR framework. The best known quality models for evaluating quality include McCall [14], Boehm [2], FURPS [9], ISO 9126 [11] and Dromey [6]. However, the term evolvability is not explicitly addressed in any of the quality models. An ontological basis which allows for the formal definition of a system and its change at the architectural level is presented in [17]. [18] proposed a taxonomy to address change as factors and classify evolvability into several aspects, e.g. generality, adaptability, scalability and extensibility. However, it does not cover all the types of software evolution, e.g. concerns of product line development.

5. Conclusions and future work

This paper proposes and demonstrates an evolvability model and an evaluation approach, which were applied into complex industrial context to assist software evolvability analysis. By establishing the evolvability model, we hope to have improved the capability in being able to on forehand understand and analyze systematically the impact of a change stimulus. This, in turn, helps us to prolong the evolution stage.

We intend to continue working on the evolvability model by conducting more case studies to confirm and refine the model. Further we plan to analyze the correlations among the subcharacteristics with respect to constraints and tradeoffs.

6. References

[1] K. Bennett and V. Rajlich. Software Maintenance and Evolution: a Roadmap. The Future of Software Engineering, Anthony Finkelstein (Ed.), ACM Press 2000.

[2] B.W. Boehm et al. Characteristics of Software Quality. Amsterdam, North-Holland, 1978.

[3] J. Bosch. Design and Use of Software Architectures – Adopting and Evolving a Product-Line Approach. Addison-Wesley. 2000.

[4] N. Chapin et al. Types of Software Evolution and Software Maintenance, Journal of Software Maintenance and Evolution: Research and Practice, 2001.

[5] S. Ciraci and P. Broek. Evolvability as a Quality Attribute of Software Architectures. The International ERCIM Workshop on Software Evolution 2006.

[6] G. Dromey. Cornering the Chimera. IEEE Software (January): 33-43, 1996.

[7] N.S. Eickelmann and D.J. Richardson. What Makes One Software Architecture More Testable Than Another? SIGSOFT Workshop, 1996.

[8] R. Fitzpatrick et al. Software Quality Challenges. 26th International Conference on Software Engineering, 2004.

[9] R. Grady and D. Caswell. Software Metrics: Establishing a Company-Wide Program. Englewood Cliffs, NJ, PrenticeHall. 1987.

[10] D. Isaac and G. McConaughy. The Role of Architecture and Evolutionary Development in Accommodating Change. Proc. NCOSE'94, 1994.

[11] ISO/IEC 9126-1. International Standard. Software Engineering. Product Quality – Part 1: Quality Model, 2001.

[12] Laprie, Dependable Computing and Fault-Tolerant Systems. Vol. 5, Dependability: Basic Concepts and Terminology. Laprie, J.C. (ed.). New York: Springer, 1992

[13] M.M. Lehman and J.F. Ramil et al. Metrics and Laws of Software Evolution – The Nineties View. IEEE Computer Press, pp 20-32, 1997.

[14] J.A. McCall, P.K. Richards and G.F. Walters. Factors in Software Quality. National Technical Information Service, 1977.

[15] T.M. Pigoski. Practical Software Maintenance. Wiley Computer Publishing, 1996.

[16] J.F. Ramil and M.M. Lehman. Metrics of Software Evolution as Effort Predictors – A Case Study. ICSM, 2000.

[17] D. Rowe and J. Leaney. Evaluating Evolvability of Computer Based Systems Architectures – an Ontological Approach. Proc. of International Conference and Workshop on Engineering of Computer-Based Systems, 1997.

[18] D. Rowe and J. Leaney. Defining Systems Evolvability – a Taxonomy of Change. Proc. of the IEEE Conference on Computer Based Systems, 1998.

[19] N. Subramanian and L. Chung. Process-Oriented Metrics for Software Architecture Evolvability. 6th IWPSE, 2002.

[20] T. Tamai and Y. Torimitsu. Software Lifetime and its Evolution Process over Generations. ICSM, 1992.

[21] N.H. Weiderman et al. Approaches to Legacy Systems Evolution. Technical Report CMU/SEI-97-TR-014, 1997.