

Software Architecture Evolution and Software Evolvability

Licentiate Thesis Proposal, September 23rd 2008

Hongyu Pei Breivold
ABB Corporate Research
hongyu.pei-breivold@se.abb.com

Mälardalen University
School of Innovation, Design and Engineering
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. For such systems, there is a need to address evolvability explicitly during the entire lifecycle, carry out software evolution efficiently and reliably, and prolong the productive life of the software systems.

In this research, we study evolution of software architecture and investigate ways to support this evolution. We focus on several particular aspects: what characteristics are necessary to constitute an evolvable software system, how to assess evolvability in a systematic manner, what needs to be considered from software maintenance and evolution perspective to address the challenges posed by the software architecture evolution and advances of technology.

Table of Contents

| | |
|-------------------------------------|-----------|
| 1. Background and Motivation | 3 |
| 2. Research Description | 3 |
| 2.1 Purpose | 3 |
| 2.2 Research Questions | 3 |
| 2.3 Methodology | 5 |
| 2.4 Contribution | 6 |
| 3. Thesis Contents | 7 |
| 3.1 Introduction | 7 |
| 3.2 State of the Art Section | 7 |
| 3.3 Methodology | 7 |
| 3.4 Contribution | 7 |
| 3.5 Publications | 7 |
| 3.6 Conclusions and Future Work | 8 |
| 4. Related Work | 9 |
| 5. Progress and Time Plan | 11 |
| 6. Full List of Publications | 12 |
| 7. References | 13 |

1. Background and Motivation

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. For such systems, there is a need to address evolvability explicitly during the entire lifecycle, carry out software evolution efficiently and reliably, 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 the software lifecycle.

As software evolvability is a fundamental element for increasing strategic and economic value of the software [41], the need for greater systems evolvability is becoming recognized. We have observed this need from various cases in industrial context, where evolvability was identified as a very important quality attribute that must be maintained. Despite this need, there are no systematic means for evaluating the evolvability of a system and thus no means of analyzing and comparing software systems in terms of evolvability. This leads to the reasons for this research: to find out what characteristics are necessary for a software system to be evolvable, how to assess and achieve software evolvability in a systematic manner, what needs to be considered from software maintenance and evolution perspective to address the challenges posed by the advances of software engineering, e.g. adoption of product line engineering and changes of implementation technology, and to investigate means for assessing quality impact quantitatively of specific quality metrics.

2. Research Description

Lehman [22] describes two views on software evolution: the *what and why* versus the *how* perspectives. The former perspective studies the nature of software evolution phenomenon and investigates its driving factor and impact. The latter perspective studies the pragmatic aspects, i.e. technology, methods and tools that provide the means to control software evolution. In this research, we focus on the *how* perspective of software evolution. In addition, we refer to the definition of *Software Evolvability* in [32]:

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

2.1 Purpose

The purpose of this research is to build up an evolvability model and to investigate ways to analyze and improve the ability to evolve software to the ever-changing requirements, software engineering advances and new technologies. The focus of the research is primarily aimed at improving software evolvability for embedded industrial systems that often have a lifetime of 10-30 years. These systems are subject to and may undergo a substantial amount of evolutionary changes, e.g. software technology changes, system migration to product line architecture, ever-changing managerial issues such as demands for distributed development, and ever-changing business decisions driven by marketing, etc.

2.2 Research Questions

This section presents the topic of the thesis in terms of research questions and summary of answers. The questions are derived from our hypothesis, and the answers are based on papers which are described further in section 3.5.

The first question intends to provide a starting point for further research:

What are subcharacteristics that are of primary importance for the evolvability of a software system?

(Q1)

The subcharacteristics that are of primary importance for the evolvability of a software system are described in paper B and C: *Analyzability, Architectural Integrity, Changeability, Extensibility, Portability, Testability and Domain-specific Attributes*. Paper B outlines a software evolvability model, 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. Additionally, paper C describes evolvability subcharacteristics based on the problems in the case of an industrial automation control system.

Given this set of subcharacteristics of evolvability, the next question relates to the assessment of the evolvability subcharacteristics:

How to assess software evolvability in a systematic manner?

(Q2)

Paper C describes our work in analyzing an industrial automation control system, driven by the need to improve its evolvability. A structured method has been proposed and piloted for analyzing evolvability at the architectural level - the ARchitecture Evolvability Analysis (AREA) method. The implications of the potential improvement strategies and evolution path of the software architecture are analyzed with respect to the evolvability subcharacteristics. The result is that the architecture requirements, corresponding design decisions, rationale and architecture evolution path have become more explicit, better founded and documented, and that the resulting documentation of refactoring improvement proposals has been widely accepted by the involved stakeholders.

Software evolvability concerns both business and technical issues [42], since the stimuli of changes related to software evolution come from both perspectives. Any change stimulus results in a collection of potential requirements that the software architecture needs to adapt to. Some examples of change stimuli are changes in environment, organization, process, technology and stakeholders' needs. These change stimuli have impact on the software system in terms of software architecture and its quality attributes. The fact that change stimuli come from both technical and business perspectives spawns two aspects that we look into, i.e. to investigate the impact of change stimuli from technology perspective as well as from business perspective. These two aspects are further expressed through the subsequent two research questions Q3 and Q4.

(1) Investigate the impact of change stimuli from technology perspective

With frequent advances in software engineering, the need to evolve software arises. Accordingly, software evolution faces different problems and challenges as new technologies are introduced. It has been witnessed that designing and implementing a large scale and complex system is a challenging task. In this thesis, we focus on two of the most well recognized software engineering paradigms coping with this challenge, i.e. component-based software engineering (CBSE) and service-oriented software engineering (SOSE). Given this, the resulting question is:

What kind of impact needs to be considered from software evolution perspective to address the challenges posed by the advances of technological paradigms, e.g. from CBSE to SOSE?

(Q3)

Taking CBSE and SOSE engineering paradigms as examples, paper A exemplifies the necessity of making analysis and exploration of both existing and emerging technologies from evolution perspective. Paper A presents a comparison framework for component-based and service-oriented software engineering from different perspectives, including key concepts and principles, development process, technology and composition. A brief discussion of reasonable utilization, combination and adaptation of the two paradigms is also outlined through looking into a set of research studies in how they have been used for improved quality attributes. The result is that since both CBSE and SOSE can co-exist in enterprise systems and complement each other [40], a good understanding of both technologies and a thorough analysis of their impacts on quality attributes will lead to more efficient combination and adaptation of these paradigms in future software development. However, a continuation of further investigations of CBSE and SOSE technologies is not within the focus of the current research.

(2) Investigate the impact of change stimuli from business perspective

One of the main difficulties of software evolution is that all artifacts produced and used during the entire software lifecycle are subject to changes [27]. Meanwhile, to keep up with new business opportunities, the need for differentiation in the marketplace, with short time-to-market as part of the need, has put critical demands on the effectiveness of software reuse. In this context, the change stimuli come from the business perspective. Accordingly, software product line approach has emerged as one specific type of software evolution, and has become one of the most established strategies for achieving large-scale software reuse and ensuring rapid development of new products [5]. However, product line development seldom starts from scratch. Instead, it is very often based on existing legacy implementations [18] and the issue of keeping legacy systems operational has become critical. Accordingly, an important and challenging type of software evolution is how to cost-effectively manage the migration of legacy systems towards product lines. This leads to the following research question:

What needs to be considered from software evolution perspective to address the challenges posed by the adoption of product line approach?

(Q4)

Paper D describes our work in two industrial cases, driven by the need to transform the existing systems towards product line architectures in order to improve evolvability. A structured product line migration method has been proposed with focus on the migration process when the migration decision has been made. In addition, applying a software product line approach to legacy systems requires that care is taken to ensure that critical aspects are considered for a smooth and successful product line migration. Through applying the migration method in the two industrial cases, observations have been made with respect to business, organization, development process and technology when adopting a product line approach. The experiences from the case studies are also described in paper D to recommend practices that proved particularly useful.

As a continuation of the first research question, one additional contribution of the thesis is a deeper study of one of the measuring attributes. A deeper study of any of the measuring attributes would be interesting and relevant, and my choice is to focus on modularity. This is motivated mainly by the fact that modularity affects the behavior of a design with respect to most of the evolvability subcharacteristics, and that not much data has been published with respect to large scale industrial software systems [20]. This leads to the following question:

What modularization means can be used to support software architecture evolution? (Q5)

Through an industrial case study in static dependency analysis, paper E explores the relationship between software evolvability, modularity and inter-module dependencies, and describes the experiences and reflections on using dependency model to support software architecture evolution. In addition, as part of the dependency analysis process, some means for providing modularization have been identified. These means can be used to support software evolution and to provide one way to let some part of a system change independently of all other parts. An additional observation is the potentials of combining different means for improved modularization and quality attributes, thus to support software evolution.

2.3 Methodology

The methodology that has been used in the research consists of the following steps:

- Analysis of the state-of-the-art and state-of-the-practice of the existing quality models for software evolution
- Analysis of the state-of-the-art of the existing process models for software evolution
- Based on the knowledge from the quality models and process models, case studies have been performed to understand subcharacteristics of the evolvability of a software system
- Case studies have been performed to investigate ways to adapt software to technology advances and the emergence of product line software development

Through the first two steps, a thorough investigation of the well-known quality models was made and the idea of a characterization of software architecture evolution was outlined. Besides, a characterization of a studied system was created in the third step. This characterization and the results from the case study are reported in paper B and C.

The third step includes also case studies with two development organizations to address the issues with software architecture evolution. The product line migration reported in paper D is based on two cases, whereas the software architecture evolution through the usage of dependency model described in paper E is based on one industrial case. The data collection for both papers was through interviews, document reviews and through the collection of data made by the practitioners. The data collection for paper A is based on literature surveys.

Four types of validity have been considered in this research study: construct validity, internal validity, external validity, and reliability [44].

Construct validity relates to the data collected and how the data represent the investigated phenomenon. It is addressed through multiple sources for the data in the project: more than one interviewee, the researchers' experiences in software product development and the usage of document reviews.

Internal validity concerns the connection between the observed behavior and the proposed explanation for the behavior. It is addressed through the reviews from several researchers.

External validity concerns the possibilities to generalize the results from a study. It is addressed through the selection of cases from two different domains and organizations (for example in paper D).

Reliability concerns the possibilities to reach the same conclusions if the study is repeated by another researcher. It is addressed through the detailed description of the procedures used and proper documentation in the case studies.

2.4 Contribution

In this thesis, we outline a software evolvability model that provides a basis for analyzing and evaluating software evolvability. This model refines software evolvability into a collection of subcharacteristics that can be measured through a number of measuring attributes. In addition, we further explore one particular measuring attribute, i.e. modularity, which affects the behavior of a design with respect to most of the evolvability subcharacteristics, as designing software for ease of extension and contraction depends on how well the software structure is organized and modular designs are argued to be more evolvable, i.e. these designs facilitate making future adaptations.

We also introduce a structured method for analyzing evolvability at the architectural level - the ARchitecture Evolvability Analysis (AREA) method that focuses on improving the capability in being able to on forehand understand and analyze systematically the impact of a change stimulus. The method is studied in an industry setting.

The fact that change stimuli come from both technical and business perspectives spawns two aspects that we focus on in the thesis, i.e. to investigate the impact of change stimuli from technology perspective as well as from business perspective. From technology perspective, we take CBSE and SOSE engineering paradigms as examples and investigate the impact of the emergence of a new engineering paradigm. We exemplify the necessity of making analysis and exploration of both existing and emerging technologies from evolution perspective. From business perspective, we focus on managing the migration of legacy systems towards product lines due to the need for differentiation in the marketplace, with short time-to-market as part of the need. Two industry cases are studied in detail. Observations have been made with respect to business, organization, development process and technology when adopting a product line approach, and the experiences from the case studies are also described to recommend practices that proved particularly useful.

The contribution of the thesis is visualized in Figure 1.

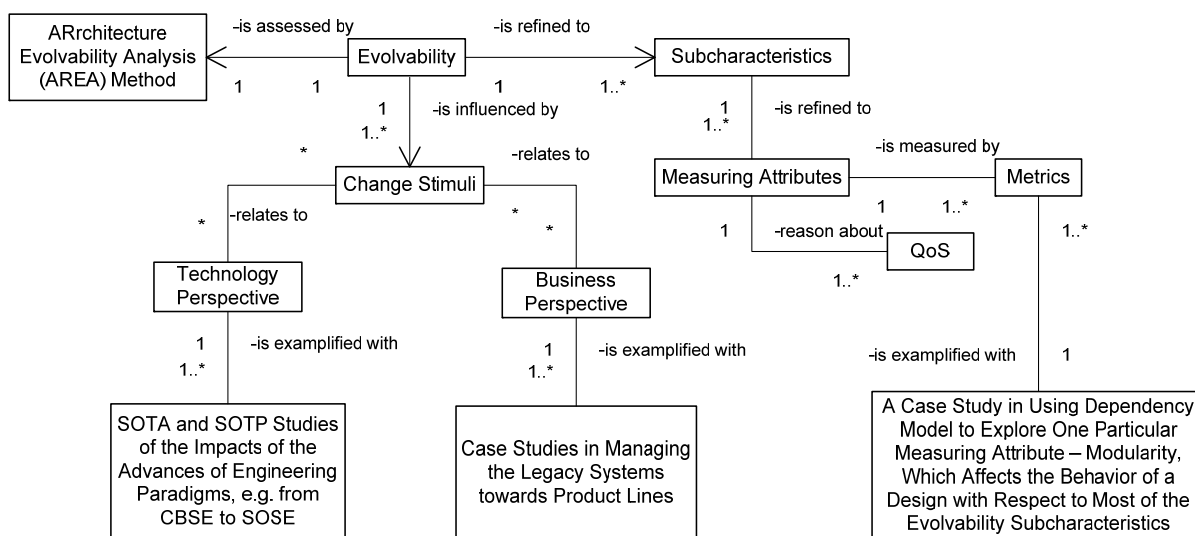


Figure 1 Contribution of the thesis

3. Thesis Contents

The thesis will be written as a collection of some published papers, together with introduction, methodology section, state of the art and practice of the research fields, and some summarizing discussion and conclusion. This will result in some duplication of information, such as brief descriptions of the same cases in several chapters, but the choice is motivated by two factors: first, the papers have been peer reviewed and published in the academic community, and second, although there are similar case descriptions in several publications, the information enables the readers to select the chapters that are of most interest to them. The following outlines the sections of the thesis.

3.1 Introduction

The introduction will present general descriptions of software evolution and describe the background and motivation to the research.

3.2 State of the Art Section

This section presents relevant fields of research and practice. This section will be an extended version of the related work section.

3.3 Methodology

A general discussion on methodology will be included, and a more thorough description of the specific methods used for the different parts of the research as outlined in section 2.3 of this proposal.

3.4 Contribution

The contribution of the thesis will be the answers to the questions formulated in section 2.2. This section will be a summary of the findings and conclusions from the included papers, and thereby, this section will also be an outline of the remainder of the thesis.

3.5 Publications

The thesis will include the following papers:

Paper A. “Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles”. Hongyu Pei Breivold, Magnus Larsson. Proceedings of the 33rd Euromicro conference on Software Engineering and Advanced Applications (SEAA), Component Based Software Engineering (CBSE) Track, IEEE, Lübeck, Germany, 2007.

This paper describes a comparison analysis framework of Component-Based Software Engineering (CBSE) and Service-Oriented Software Engineering (SOSE), and analyzes them from a variety of perspectives. We discuss as well the possibility of combining the strengths of the two paradigms to meet non-functional requirements. This paper clarifies the characteristics of CBSE and SOSE, tries to shorten the gap between them and bring the two worlds together so that researchers and practitioners become aware of essential issues of both paradigms, which may serve as inputs for further utilizing them in a reasonable and complementary way.

I was the main author and contributed with the comparison analysis framework, the analysis and conclusions. The coauthor contributed with advice and discussions regarding the analysis, conclusions and reviews. In addition, my supervisor professor Ivica Crnkovic contributed with valuable feedback and comments through reviews.

Paper B. “Analyzing Software Evolvability”. Hongyu Pei Breivold, Ivica Crnkovic, Peter Eriksson. Proceedings of the 32nd IEEE International computer Software and Applications Conference (COMPSAC), Turku, Finland, July, 2008.

This paper describes 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.

I was the main author and contributed with the proposed evolvability model and the case study. The coauthors contributed with advices regarding the methodology, discussions regarding the analysis, conclusions and reviews.

Paper C. “Analyzing Software Evolvability of an Industrial Automation Control System: A Case Study”. Hongyu Pei Breivold, Ivica Crnkovic, Rikard Land, Magnus Larsson. Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA), IEEE, Sliema, Malta, October, 2008.

This paper describes our work in analyzing and improving the evolvability of an industrial automation control system, and presents 1) evolvability subcharacteristics based on the problems in the case and available literature; 2) a structured method for analyzing evolvability at the architectural level - the ARchitecture Evolvability Analysis (AREA) method. This paper includes also the main analysis results and our observations during the evolvability analysis process in the case study.

I was the main author and contributed with the description of the proposed method, the case study, the analysis and conclusions. The coauthors contributed with advice regarding methodology, discussions regarding the analysis and reviews.

Paper D. “Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies”. Hongyu Pei Breivold, Stig Larsson, Rikard Land. Proceedings of the 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Software Process and Product Improvement (SPPI) Track, IEEE, Parma, Italy, September, 2008.

This paper presents a structured migration method and describes our experiences in migrating industrial legacy systems into product lines. In addition, we present a number of specific recommendations for the transition process which will be of value to organizations that are considering a product line approach to their business. The recommendations cover four perspectives: business, organization, product development processes and technology.

I was the main author and contributed with the description of good practices in product line migration, the analysis and conclusions. The coauthors contributed with advice regarding methodology and reviews.

Paper E. “Using Dependency Model to Support Software Architecture Evolution”. Hongyu Pei Breivold, Ivica Crnkovic, Rikard Land, Stig Larsson. Proceedings of the 4th International ERCIM Workshop on Software Evolution and Evolvability (Evol’08), IEEE, L’Aquila, Italy, September, 2008.

This paper explores the relationships between evolvability, modularity and inter-module dependency, as designing software for ease of extension and contraction depends on how well the software structure is organized. Through a case study of an industrial power control and protection system, we describe our work in managing its software architecture evolution, guided by the dependency analysis at the architectural level. The paper includes also the main analysis results, our experiences and reflections during the dependency analysis process in the case study.

I was the main author and led the study. I contributed with the description of managing software evolution using the dependency analysis results as inputs, as well as the analysis and conclusions. The coauthors contributed with advice regarding case description and reviews.

3.6 Conclusions and Future Work

This section will summarize the thesis and propose areas for future research.

4. Related Work

This section describes work that has been done related to software evolution, system and software architecture and software quality.

The seminal papers within software evolution are [23, 24], in which Lehman formulated eight laws of software evolution based on the observations of the IBM OS/360 operating system, and deliberately used the term software evolution to address the difference with the post-deployment activity of software maintenance.

Several process models have been proposed since the late seventies. For instance, Yau [43] proposed in the seventies the evolutionary process model with change mini-cycle, in which important new activities such as change impact analysis and change propagation are identified to accommodate the fact that software changes are rarely isolated. In the nineties, the term software evolution gained widespread acceptance. Several process models have emerged, e.g. Gilb's evolutionary development [13], Bohem's spiral model [7] and Bennett and Rajlich's staged model [3].

Mira Kajko-Mattsson created two process models of problem management within corrective maintenance CM3: Back-End Problem Management and CM3: Front-End Problem Management. Besides, SYSLAB, the Information Systems Laboratory (<http://syslab.dsv.su.se/>) is in the process of developing a comprehensive process model for industrial evolution and maintenance. The model is called Evolution and Maintenance Management Model (EM3). It consists of the following constituent models: Process Models within Corrective Maintenance (CM3) which consist of Front-End Problem Management, Back-End Problem Management and Emergency Problem Management, as well as process models within Evoluton (EM3) which consist of SLA Management, Education and Training, Predelivery/Prerelease Maintenance and Release Management.

As the importance of evolvability is being recognized, several metrics have been proposed for evaluating evolvability. Ramil and Lehman proposed metrics based on implementation change logs [31] and computation of metrics using the number of modules in a software system [21]. Another set of metrics is based on software life span and software size [38]. In [35], 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 [10].

A general description of different quality models can be found in [29]. In quality models, quality attributes are decomposed into various factors, leading to various quality factor hierarchies. Some well-known quality models are McCall's quality model [26], Dromey's quality model [12], Bohem's quality model [6], ISO 9126 [15] and FURPS quality model [14].

The foundation for any software system is its architecture, which allows or precludes nearly all of the quality attributes of the system. Accordingly, several architecture analysis methods have emerged. A general description of different architecture analysis methods can be found in [11].

A specific type of software evolution is the adoption of product line architecture. Within the area of software product line evolution, Bosch [8] proposed methods for designing software architecture, in particular product line architecture. Pohl et al. [30] elaborated two key principles behind software product-line engineering: (i) separation of software development in domain and application engineering, and (ii) explicit definition and management of variability of the product line across all development artifacts. A four-dimensional software product family engineering evaluation model is described in [39] to determine the status of software family engineering concerning business, architecture, organization and process. Bayer et al [2] presents the RE_MODEL method to integrate reengineering and product line activities to achieve a transition into product line architecture. A key element in the method is the *blackboard*, a work space which is shared for both activities that are done in parallel. The PuLSETM method [33] addresses the different phases of product line development and is used to systematically analyze a component and to improve its reusability as well as maintainability. The focus was on one component enabling reuse of that component. In order to evaluate the potential of creating a product line from existing products, MAP (Mining Architectures for Product Lines) is described in [34], which focuses on the feasibility evaluation process of the organization's decision to move towards a product line. Options Analysis for Reengineering [4] is another method for mining existing components for a product line. [25] describes combining reference architecture and configuration architecture to describe legacy product family architecture and manage its evolution.

Various techniques have emerged to assess quality impact qualitatively or quantitatively of specific quality metrics. They differ from each other in terms of principles, concepts and analysis capabilities. For instance, Kataoka et al. [17] propose coupling metrics to measure the maintainability enhancement effect of a program

refactoring. Tahvildari and Kontogiannis [36] propose a re-engineering transformation framework using soft goal graph to correlate non-functional requirements with design patterns to guide transformation process. The soft goals that are refined from maintainability include coupling, cohesion, modularity, encapsulation, complexity, consistency and reuse. Another framework proposed by Tahvildari and Kontogiannis [37] combines using metrics for quality estimation and performing transformation based on soft goal graphs.

Designing and implementing a large scale and complex system is a challenging task. In this thesis, we focus on two of the most well recognized software engineering paradigms coping with this challenge, i.e. component-based software engineering (CBSE) and service-oriented software engineering (SOSE). While CBSE is an established approach in many engineering domains, SOSE has evolved from CBSE frameworks and object oriented computing to face the challenges of open environments. Because of the diverse nature of software systems, it is unlikely that systems will be developed using a purely service or component-based approach [19]. Therefore, the ability to combine the strength of CBSE and SOSE and use them in a complementary manner becomes essential. So far, a lot of research has been done in combining the strength of CBSE and SOSE for improved quality attributes of software solutions. Jiang and Willey proposed a multi-tiered architecture [16] that offers flexible and scalable solutions to the design and integration of large and distributed systems, where the architecture makes use of both services and components as architectural elements, offering flexibility and scalability in large distributed systems and meanwhile remaining the system performance. Wang and Fung [36] proposed an idea of organizing enterprise functions as services and implementing them as component-based systems in order to offer flexible, extensible and value-added services. Cervantes and Hall [9] addressed introducing service-oriented concepts into component model to provide support for late binding and dynamic component availability in component models. [28] explores how an service oriented architecture impacts different quality attributes, identifying issues and tradeoffs related to them.

5. Progress and Time Plan

The progress of the current research is:

- **Publications.** Papers A, B, C and D have been accepted at international conferences. Paper E has been accepted at an international workshop.
- **Courses.** I already fulfill the other main part required for the licentiate degree, namely completed courses with 41 credits as shown in the table below.

| Courses | Credits | Status |
|--|---------|-----------|
| UML, Object-oriented analysis and design with UML | 3 | Completed |
| Advanced C++ programming | 1 | Completed |
| COM and ActiveX | 1 | Completed |
| Software Architecture for industrial systems | 3 | Completed |
| ABB project design and analysis work | 7 | Completed |
| Research methodology | 5 | Completed |
| PROGRESS: techniques and technologies | 5 | Completed |
| Legacy issues in industrial software development | 5 | Completed |
| Advanced CBSE | 5 | Completed |
| Formal languages, automata and theory of computation | 3 | Completed |
| Research planning | 3 | Completed |

The goal is to have the licentiate seminar in January, 2009. To complete the thesis, the main remaining activities are:

- **Writing and compiling thesis.** Part of the thesis is already written in the form of papers. The other part of the thesis will be written based on the thesis contents as outlined in section 3, including an introduction, state-of-the-art section, methodology, contribution and conclusion.

A preliminary time plan is as follows:

| Number | Task | Start | End | Duration | Q3 - 2008 | | | Q4 - 2008 | | Q1 - 2009 | | | |
|--------|--|------------|------------|----------|-----------|--------|-----------|-----------|----------|-----------|---------|----------|-------|
| | | | | | July | August | September | October | November | December | January | February | March |
| 1 | Thesis Proposal Presentation | 9/23/2008 | 9/24/2008 | 1 | | | | | | | | | |
| 2 | Write First Draft of the Thesis | 9/1/2008 | 10/15/2008 | 32 | | | | | | | | | |
| 3 | Complete thesis and Send to Opponent | 10/16/2008 | 11/15/2008 | 22 | | | | | | | | | |
| 4 | Finish Camera-Ready Copy of the Thesis | 11/15/2008 | 12/15/2008 | 20 | | | | | | | | | |
| 5 | Licentiate seminarium | 1/15/2009 | 1/16/2009 | 1 | | | | | | | | | |

Potential opponents are Patricia Lago (Vrije University), Zeljka Car (FER, Zagreb University) and Mira Kajko-Mattsson (Stockholm University).

6. Full List of Publications

Conferences and workshops

- Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles
Hongyu Pei Breivold, Magnus Larsson
33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track, IEEE, Lübeck, Germany, August, 2007
- Evaluating Software Evolvability
Hongyu Pei Breivold, Ivica Crnkovic, Peter Eriksson
7th Conference on Software Engineering and Practice in Sweden (SERPS), Göteborg, Sweden, October, 2007
- Analyzing Software Evolvability
Hongyu Pei Breivold, Ivica Crnkovic, Peter Eriksson
32nd IEEE International computer Software and Applications Conference (COMPSAC), Turku, Finland, July, 2008
- Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies
Hongyu Pei Breivold, Stig Larsson, Rikard Land
34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Software Process and Product Improvement (SPPI) Track, IEEE, Parma, Italy, September, 2008
- Using Dependency Model to Support Software Architecture Evolution
Hongyu Pei Breivold, Ivica Crnkovic, Rikard Land, Stig Larsson
4th International ERCIM Workshop on Software Evolution and Evolvability (Evol'08), IEEE, L'Aquila, Italy, September, 2008
- Analyzing Software Evolvability of an Industrial Automation Control System: A Case Study
Hongyu Pei Breivold, Ivica Crnkovic, Rikard Land, Magnus Larsson
3rd International Conference on Software Engineering Advances (ICSEA), IEEE, Sliema, Malta, October, 2008

MRTC Report

- Using Software Evolvability Model for Evolvability Analysis
Hongyu Pei Breivold, Ivica Crnkovic
MRTC report ISSN 1404-3041 ISRN MDH-MRTC-222/2008-1-SE, Mälardalen Real-Time Research Center, Mälardalen University, February, 2008

Tutorial

- Emerging Technologies in Industrial Context – Component-Based and Service-Oriented Software Engineering
Ivica Crnkovic, Hongyu Pei Breivold
31st IEEE International computer Software and Applications Conference (COMPSAC), Beijing, China, July, 2007

7. References

- [1] Bass L., Clements P., and Kazman R., *Software Architecture in Practice* (2nd edition), ISBN 0-321-15495-9, Addison-Wesley, 2003.
- [2] Bayer, J. Girard, J. F., Würthner, M., DeBaud, J. M. and Apel, M., “*Transitioning legacy assets to a product line architecture*”, Proceedings of the 7th European Software Engineering Conference, Toulouse, France: Springer, 1999.
- [3] Bennett, K. and Rajlich, V., “*Software Maintenance and Evolution: a Roadmap*”, the Future of Software Engineering, ACM Press, 2000.
- [4] Bergey, J., O'Brien, L. and Smith, D., “*Using options analysis for reengineering (OAR) for mining components for a product line*”, Proceedings of Second Software Product Line Conference, volume 2379, pp. 316-327, Springer, 2002.
- [5] Birk, A., Heller, G., John, J., Schmid, K. et al., “*Product Line Engineering: The State of the Practice*”, IEEE Software, 2003.
- [6] Boehm, B. W. et al., *Characteristics of Software Quality*, Amsterdam, North-Holland, 1978.
- [7] Boehm, B. W., *A Spiral Model of Software Development and Enhancement*, IEEE Computer, 1988.
- [8] Bosch, J., *Design and use of software architectures: adopting and evolving a product-line approach*, ACM Press/Addison-Wesley Publishing Co., 2000.
- [9] Cervantes, H. and Hall, R. S., “*Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model*”, 2004.
- [10] Chung, L. et al, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.
- [11] Dobrica, L. and Niemela, E., “*A Survey on Software Architecture Analysis Methods*”, IEEE Transactions on Software Engineering, vol. 28, No. 7, pp. 638-653, 2002.
- [12] Dromey, G., “*Cornering the Chimera*”, IEEE Software (January): 33-43, 1996.
- [13] Gilb, T., *Evolutionary Development*, ACM Software Engineering Notes, 1981.
- [14] Grady, R. and Caswell, D., *Software Metrics: Establishing a Company-Wide Program*, Englewood Cliffs, NJ, PrenticeHall, 1987.
- [15] ISO/IEC 9126-1, International Standard, Software Engineering. Product Quality – Part 1: Quality Model, 2001.
- [16] Jiang, M. and Willy, A., “*Architecting Systems with Components and Services*”, Information Reuse and Intergration, 2005.
- [17] Kataoka, Y. et al., “*A Quantitative Evaluation of Maintainability Enhancement by Refactoring*”, Proceedings of International Conference on Software Maintenance, 2002.
- [18] Kotonya, G. and Hutchinson, J., “*A Component-Based Process for Modelling and Evolving Legacy Systems*”, Software Process: Improvement and Practice, 13(2), pp. 113-125, 2008.
- [19] Kotonya, G. Hutchinson, J. and Bloin, B., “*A Method for Formulating and Architecting Component and Service-Oriented Systems*”, http://scse.comp.lancs.ac.uk/pubs/KotonyaHutchinsonBloin_SOSEBook.pdf, visited 2007.
- [20] LaMantia, M. J., Cai, Y. et al., “*Analyzing the Evolution of Large-Scale Software Systems using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases*”, WICSA, 2008.
- [21] Lehman, M. M. and Ramil, J. F. et al., “*Metrics and Laws of Software Evolution – The Nineties View*”, IEEE Computer Press, pp 20-32, 1997.
- [22] Lehman, M., Ramil, J. F. and Kahen, G., “*Evolution as a Noun and Evolution as a Verb*”, Proceedings of Workshop on Software and Organization Co-Evolution (SOCE), 2000.
- [23] Lehman, M. M., “*Programs, Life Cycles, and Laws of Software Evolution*”, Proceedings of IEEE 68(9), 1980.
- [24] Lehman, M. M., “*On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle*”, Systems and Software, 1980.
- [25] Maccari, A. and Riva, C., “*Architectural evolution of legacy product families*”, Proceedings of the Fourth International Workshop on Product Family Engineering, 2001.
- [26] McCall, J. A., Richards, P. K. and Walters, G. F., *Factors in Software Quality*, National Technical Information Service, 1977.
- [27] Mens, T. and Demeyer, S., *Software Evolution*, ISBN 978-3-540-76439-7, Springer, 2008.
- [28] O'Brien, L., Merson, P. and Bass, L., “*Quality Attributes for Service-Oriented Architectures*”, Proceedings of the 29th International Conference on Software Engineering, 2007.

-
- [29] Ortega, M., Perez, M. and Rojas, T., “*Construction of a Systemic Quality Model for Evaluating a Software Product*”, Software Quality Journal, 2003.
 - [30] Pohl, K., Böckle, G. and van der Linden, F., *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer, 2005.
 - [31] Ramil, J. F. and Lehman, M. M., “*Metrics of Software Evolution as Effort Predictors – A Case Study*”, Proceedings of the International Conference on Software Maintenance, 2000.
 - [32] Rowe, D. and Leaney, J., “*Defining Systems Evolvability – a Taxonomy of Change*”, Proceedings of the IEEE Conference on Computer Based Systems, 1998.
 - [33] Schmid, K., John, I., Kolb, R. and Meier, G., “*Introducing the PuLSE Approach to an Embedded System Population at Testo AG*”, ICSE, 2005.
 - [34] Stoermer, C. and O'Brien, L., “*MAP - mining architectures for product line evaluations*”, Proceedings of WICSA'01, pages 35-44, IEEE Computer Society Press, Aug. 2001.
 - [35] Subramanian, N. and Chung, L., “*Process-Oriented Metrics for Software Architecture Evolvability*”, Proceedings of the 6th International Workshop on Principles of Software Evolution, 2002.
 - [36] Tahvildari, L and Kontogiannis, K., “*A Methodology for Developing Transformations Using the Maintainability Soft-Goal Graph*”, Proceedings of 9th Working Conference on Reverse Engineering, 2002.
 - [37] Tahvildari, L and Kontogiannis, K., “*A Metric-Based Approach to Enhance Design Quality through Meta-Pattern Transformations*”, Proceedings of 7th European Conference on Software Maintenance and Reengineering, 2003.
 - [38] Tamai, T. and Torimitsu, Y., “*Software Lifetime and its Evolution Process over Generations*”, Proceedings of the International Conference on Software Maintenance, 1992.
 - [39] van der Linden, F., Bosch, J., Kamsties, E., Kansala, K. and Obbink, H., “*Software Product Family Evaluation*”, Proceedings of SPLC, 2004.
 - [40] Wang, G. and Fung, C. K., “*Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems*”, Proceedings of the 37th Hawaii International Conference on Systems Sciences, 2004.
 - [41] Weiderman, N.H. et al., “*Approaches to Legacy Systems Evolution*”, Technical Report CMU/SEI-97-TR-014, 1997.
 - [42] Yang, H. and Ward, M., *Successful Evolution of Software Systems*, Artech House Publishers, London, 2003.
 - [43] Yau, S. S., Colofello, J. S. and MacGregor, T., “*Ripple Effect Analysis of Software Maintenance*”, Proceedings of COMPSAC, 1978.
 - [44] Yin, R. K., *Case Study Research: Design and Methods*, ISBN 0-7619-2553-8, Sage Publications, 2003.