Research Planning Assignment 3
Research Overview

Hongyu Pei Breivold
Department of Computer Science and Electronics
Märlardalen University, Sweden
hongyu.pei-breivold@mdh.se
1. Research Area Overview

1.1 Research Area

My research project is called Architecting Software Evolution.

Software evolution is an important issue in software engineering. It is concerned with modifying software once it is delivered to a customer. Very often the term software maintenance is used as equality to software evolution. According to the IEEE definition, software maintenance is the process of modifying the software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a change in environment.

During the evolution of these software intensive systems, reducing system complexity and improving software quality are the most after striving goals. However, most of the software intensive systems nowadays become more and more complex due to the constantly incoming new functionality and business requirements and evolution of technologies.

Leintz and Swanson categorize maintenance into four categories [9]:

(1) Perfective maintenance – changes required as a result of user request
(2) Adaptive maintenance – changes needed as a consequence of environment change
(3) Corrective maintenance – the identification and removal of faults in the software
(4) Preventive maintenance – changes made to software to make it more maintainable

Consequently, as the software is enhanced, modified and adapted to meet new functionality and business requirements as well as emerging technology demands during the system’s life cycle, the system becomes more complex, leading to declined system quality. Therefore, while designing and implementing a large scale and complex system has been a challenging task, evolving and maintaining the software system to reduce complexity and meanwhile meet quality attribute requirements during the system life cycle has become even more challenging.

The focus areas of the research will include from the software architecture perspective, how to model the system, which methods and tools can be used and how to verify that the system still can achieve satisfactory quality attributes during the evolution process. The systems that will be analysed belong to a category of complex industrial embedded systems and industrial automation systems. The work will focus on particular quality attributes related to maintainability, portability, and evolvability. Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. Portability is the ability of the system to run under different computing environments. These environments can be hardware, software or a combination of the two. Evolvability is the ability to withstand changes in requirements, environment and implementation technologies. Model-based and component-based approaches will be utilised to support software architectural analysis and modelling of the system.

Based on the literature survey that I have done so far, several methods for assessing quality properties, such as maintainability, are worth further investigation. General descriptions about these methods will be covered in the related work section. These methods assess quality impact either qualitatively or quantitatively in terms of specific quality metrics. They differ from each other in terms of principles, concepts and analysis capabilities as well. It would not be easy to directly apply them into a software-
intensive system within industrial context. Therefore, an idea is to try to extract certain parts of the methods and combine them to some extent to establish a framework and apply to a case study in industrial context. Thereafter, an analysis will be made from several perspectives, such as lessons learned, any suggestions for improvement of the method, any limitations, and validation of the results, etc. Hopefully, the revised framework will be applied further to a new case study.

1.2 Research Method

The research will be based on literature surveys and case studies and belongs to the domain of empirical research. The research problems will be derived from industrial problems. Theories and methods that are to be proposed in the research will be applied on particular cases from the industry. The methods and the results from the case study will be analysed. Thereafter, the methods will be refined and applied further to new cases. Theoretical studies and validation of the methods will be emphasised. The improved methods from software architectural analysis and system modelling perspective will be proposed and validated. All research will be close to industrial cases, i.e. the research is more of the applied research type then fundamental.

2. Current Research Issues and Hot Topics

In the past few years, we have witnessed an enormous expansion in the use of software in business, research, industry, and critical infrastructure systems. While new software and systems are developed at tremendous speed, they become legacy systems at fast speed too. Several factors contribute to this phenomenon.

(1) Software aging

Software ages quickly because the overall business requirements change at tremendous speed. Therefore, in order to survive the competition and maintain a leading position among competitors, the software systems need to keep up to the changing demands from the customers to meet new functionality requirements and to overcome any existing system limitations. The inevitable aging of all software systems have turned even rather new object oriented systems into legacy systems [4]. This factor corresponds well with Lehman’s Laws of software evolution regarding continuing change, i.e. an E-type [8] program that is used must be continually adapted, and else they become progressively less satisfactory [6].

(2) Increasing software complexity

Most of the software intensive systems nowadays become more and more complex due to the constantly incoming new requirements and evolution of technologies. It is quite common that as an evolving program is continually changed, its structure deteriorates [10], especially when we have too tight schedule to consider and analyze the consistency of design and have to fix problems before deadline and within budget. Consequently, complexity increases unless work is done to maintain or reduce it [7].

(3) Declining software quality

Unless rigorously adapted to take into account changes in the operational environment, the quality of an E-type system will appear to decline as it is evolved [6]. The extreme time-to-market pressure contributes to the degradation of software in the sense that quick fixes are done in the code without considering the potential impact of code change to the program structure and software architecture.
The knowledge and experience of software developers can influence a lot to the outcome of software system evolution as well. Developers need to have deep understanding and knowledge about software architecture, quality attributes, developing software-intensive systems and modern software engineering techniques. Misuses of object-oriented principles such as encapsulation, inheritance, etc. also result in the declining software quality.

Meanwhile, poor documentation leads to lack of understanding of the software, resulting in declined software quality.

To summarize, as the software is enhanced, modified and adapted to meet new requirements, the system becomes more complex. Therefore, while designing and implementing a large scale and complex system has been a challenging task, evolving and maintaining the software system to reduce complexity and meanwhile meet quality attribute requirements during the system life cycle has become even more challenging. Consequently, to analyze and meet quality requirements during software evolution process has become an important research issue in software engineering. During the evolution of software intensive systems, reducing system complexity and improving software quality are the most after striving goals.

The hot topics within software evolution area and quality analysis during software evolution process include but are not limited to:

(1) How can software be designed so that it can easily be evolved [1]?

(2) Are there any more effective tools and methods for program comprehension [1]?

(3) What is a better formalism and conceptualisation of maintainability [1]?

(4) How do we measure maintainability?

(5) How to model and analyze the system from software architecture perspective?

(6) Which methods and tools can be used?

(7) How to verify that the system still can achieve satisfactory quality attributes during evolution process?

(8) What are the techniques and frameworks for the identification and selection of software transformations to achieve target requirements or objectives [15]?

3. Central Literature and Seminal Papers

The research in architecting software evolution covers several areas, including software evolution, quality models, software architecture analysis and specific software quality analysis.

3.1 Software evolution area

The seminal papers within software evolution area are written by Lehman [2, 3, 4]. They are invaluable sources for understanding software evolution. In these papers, he formulated eight laws of software evolution based on the observations on the evolution of the IBM OS/360 operating system and other observations later on.
3.2 Quality model area

Quality attributes can be decomposed into various factors leading to various quality factor hierarchies commonly referred to as quality models, such as McCall’s quality model, Dromey’s quality model [21], Bohem’s quality model [22, 23], ISO 9126 [24] and IEEE 1061. [25] gives a good summary of different quality models.

3.3 Software architecture analysis area

The seminal paper within software architecture analysis area is [26]. Generally, literatures that are concerned with analyzing the relationship between software architecture and quality attributes are also of interest because they can assist in analyzing the software quality of the migrant system. For example, the Software Engineering Institute’s work in Attribute-Based Architecture Style (ABAS) correlates software architecture and quality attributes [19].

3.4 Quality analysis area

L. Tahvildari et al. wrote a paper titled quality-driven software re-engineering [14]. This paper presents a framework that allows specific NFR (non-functional requirement) such as performance and maintainability to guide the re-engineering process. Such requirements for the migrant system are modelled using soft goal interdependency graphs and are associated with specific software transformations. An evaluation procedure at each transformation step determines whether specific qualities for the new migrant system can be achieved. This paper has become the base for a couple of other frameworks (will be explained in related work section) for assessing quality properties during software evolution process.

4. Related Work

The research in architecting software evolution covers several areas, including quality models, software architecture analysis methods and specific software quality analysis methods.

4.1 Quality models

4.1.1 McCall’s quality model

This model points out three working areas in systems: product operation, product revision and product transition. The main contribution of the model is the relationship created between quality characteristics and metrics.

4.1.2 Boehm’s quality model

This quality model is similar to McCall’s quality model in the sense that it represents a hierarchical structure of characteristics, each of which contributes to total quality. This model starts with software’s general utility from various dimensions. General utility is broken down into portability, utility and maintainability. Maintainability is in turn broken down into testability, understandability and modifiability.
4.1.3 Dromey’s quality model
This model seeks to increase understanding of the relationship between the attributes (characteristics) and the sub-attributes (sub-characteristics) of quality. It also attempts to pinpoint the properties of the software product that affect the attributes of quality.

4.1.4 ISO 9126 quality model
This standard defines a framework for evaluating software product quality based on six quality attributes (called quality characteristics in ISO 9126). Quality is defined as the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. The main terminology in the quality model is software characteristic, sub-characteristic and quality attribute. This model indicates that some component of the software quality must be described in terms of one or several of six characteristics: functionality, usability, maintainability, reliability, portability and efficiency. Take maintainability as an example, it is defined with the associated sub-characteristics, i.e. analyzability, changeability, stability and testability. Portability attribute is associated with adaptability, installability, conformance and replaceability. The disadvantage of this model is not showing very clearly how these aspects can be measured.

4.1.5 IEEE 1061 quality model
Software quality is defined as the degree to which software possesses a desired combination of quality attributes. Quality attribute is defined as characteristic of software or a generic-term applying to quality factors, quality sub-factors or metric values.

4.1.6 FURPS quality model
This model takes into account five characteristics: functionality, usability, reliability, performance and supportability. The disadvantage of the model is that it fails to take account of the software product’s portability.

4.2 Software architecture analysis methods
The foundation for any software system is its architecture, which allows or precludes nearly all of the quality attributes of the system. Various methods have emerged for software architecture evaluation. The fundamental assumption is that architectural decisions determine a system’s quality attributes. Thus, it is possible to evaluate architectural decisions with respect to their impact on those attributes.

4.2.1 ATAM (Architecture Trade-off Analysis Method)
ATAM has the flexibility to analyze any quality attribute. The outputs from ATAM are prioritized quality attribute scenarios, mapping of approaches to quality attributes, risks and non risks. Besides, a catalogue of architectural approaches used, sensitivity points and trade-off points are as well the evaluation output. Information and insights about the architecture is obtained, although it is not used for precisely characterizing any quality attribute.

4.2.2 SAAM (Software Architecture Analysis Method)
SAAM concentrates on modifiability in its various forms, such as portability, subsetability and variability. The outputs from SAAM are prioritized quality attribute scenarios, mapping of approaches to quality attributes, risks and non risks. Information and insights about the architecture is obtained, although it is not used for precisely characterizing any quality attribute.
4.2.3 SAAMER (Software Architecture Analysis Method for Evolution and Reusability)

SAAMER concentrates on evolution and reusability. Evolution integrates quality objectives such as maintainability and modifiability obtained from domain experts.

4.2.4 SBAR (Scenario-Based Architecture Reengineering)

This method estimates the potential of the designed architecture to reach the software quality requirements.

4.2.5 NFR Framework

Chung [3] proposed a process-oriented framework for evaluating software architecture. In the approach, non-functional requirements are represented as goals to be addressed and achieved during the process of architectural design.

4.3 Software quality analysis methods

As stated earlier, software quality analysis during software evolution is an important issue in software engineering. During the evolution of software intensive systems, reducing system complexity and improving software quality are the most after striving goals. Consequently, the ability to estimate and measure quality characteristics of the system has become critical. Various techniques have emerged and they assess quality impact either qualitatively or quantitatively in terms of specific quality metrics. They differ from each other in terms of principles, concepts and analysis capabilities as well.

The first section describes the relevance of related work to my research. The rest of the section describes briefly the related work

4.3.1 Relevance to research

The first three techniques that will be presented in related work section have very limited capabilities to be applied into complicated software intensive systems in industrial context. The first one deals only with coupling aspect in maintainability issue. The second and third methods correlate design patterns and architecture quality analysis. This brings up some limitations in the sense that a variety of factors can have effects on software quality. Design patterns are only part of the many factors.

Therefore, the fourth and the fifth methods can be of interest for further investigation. They have different approaches in the sense that the fourth method uses scenarios while the fifth method is based on the Non-Functional Requirement (NFR) framework.

4.3.2 Coupling metrics – 1st method

Kataoka et al. propose coupling metrics as a quantitative evaluation method to measure the maintainability enhancement effect of a program refactoring [5].

There are various aspects of maintainability of a program, such as coupling, cohesion, size and complexity, description, etc. All of them contribute to the maintainability quality attribute of the software system in the sense that low coupling and high cohesion modules enhance system maintainability, simple and small modules are easy to maintain, appropriate naming rules help with program understanding.

Among the above mentioned aspects, coupling is selected as a maintainability quantification metrics and classified into three categories, i.e. return value coupling, parameter coupling and shared variable coupling. Coefficients for respective coupling need to be defined in order to represent the impact
degree of respective coupling in a method refactoring. Thereafter, measurement of the metrics is made before and after the refactoring and comparison becomes explicit, with the delta value indicating the maintainability enhancement effect.

The application of this evaluation method demands extensive experiences in making good judgement on finding appropriate coefficient values to calculate the coupling metrics value.

4.3.3 Transformation using soft goal graph – 2nd method

Tahvildari and Kontogiannis propose a re-engineering transformation framework for object oriented legacy systems. This transformation framework correlates non-functional requirements with design patterns to guide transformation process [12]. The definition and refinement of quality requirement is based on NFR framework [3], where a soft goal interdependency graph is used to support modelling of design rationale.

Because the use of design patterns has impact on system quality attributes, the goal of the transformation framework is to associate each design pattern transformation to one or more soft goals.

The transformation framework has the feasibility to analyze any quality attributes of a software system though Tahvildari and Kontogiannis focused on maintainability as an illustrating example. The soft goals that are refined from maintainability are:

2. Cohesion: high cohesion modules are easy to maintain.
3. Modularity: programs that have many direct interrelationships between any two random parts of the program code are less modular than programs where those relationships occur mainly at well-defined interfaces between modules [17]. High modularity enhances system maintainability.
4. Encapsulation: a software module hides information by encapsulating the information into a module that are most likely to change, thus protecting other parts of the program from changing [16] and enhancing system maintainability.
5. Complexity: low I/O complexity modules are easy to maintain.
6. Consistency: high control flow consistency and high data consistency enhance system maintainability.
7. Reuse: high module reuse enhances system maintainability.

Six categories of primitive design pattern transformations are identified, such as Abstraction, Extension, Movement, Encapsulation, etc. and they can be combined to produce complex design pattern transformations related to various design patterns. Thereafter, a qualitative association (positive or negative impact) is identified between each design pattern transformation and the above mentioned soft goals, i.e. the aspects of maintainability.

4.3.4 Metric-based transformation – 3rd method

Tahvildari and Kontogiannis propose also another framework for detection and correction of design defects on class level in object oriented legacy systems [13].

A catalogue of object-oriented metrics is used as indicators for automatically detecting where a particular transformation can be applied to improve the software quality. The object oriented metrics
are classified into three categories: complexity metrics, coupling metrics and cohesion metrics. Examples of these object oriented metrics are:

1. CDE (Class Definition Entropy)
2. NOM (Number of Methods)
3. WMC (Weighted Methods per Class) in complexity category
4. DAC (Data Abstraction Coupling)
5. RFC (Response For a Class) in coupling category
6. LCOM (Lack of Cohesion in Methods)
7. TCC (Tight Class Cohesion) in cohesion category.

The detection process in the framework includes checking design principles and detecting violations by using different design heuristics [11], such as key classes, one class – one concept. The correction process in the framework is based on analyzing the impact of meta-pattern transformations, extracted from the previous approach (transformation using soft goal graph), on these object oriented metrics.

When classes that need refactoring are detected by applying certain design heuristics, suitable meta-pattern transformation can be selected to solve the violation problem to design heuristics, thus to enhance system maintainability.

This approach combines using metrics for quality estimation and performing transformation based on soft goal graphs. It analyzes the interaction between software transformations and metrics.

4.3.5 Mapping from quality goals to analytic models -4th method

The researchers at Software Engineering Institute (SEI) presented a sequence of steps that map architectural quality goals into scenarios that measure the goals, mechanisms that realize the scenarios and analytic models that measure the results [2].

The first step in this method is to define the generic quality goals, such as performance, availability and modifiability. Then the desired qualities are mapped to abstract and derived/specific scenarios. The next step is to propose a mechanism for responding to the abstract and derived/specific scenarios. Once a number of scenarios and a mechanism for satisfying them have been chosen, an associated analytic model must be built. Because the process of architectural design and analysis is very complex and involves many competing factors, it is essential to create analytic models for all attributes of interest in all risk areas. For example, rate monotonic analysis (RMA) is used for performance analysis and Markov modelling is used for availability analysis.

One interesting aspect that is worth mentioning here in the method is that one does not build a model for every part of a system. Models are only built for areas of the system that appear to be potential risks. During the software evolution process, the analytic model can be used to assess the impact of architectural changes, relative to the system’s changing quality goals.

4.3.6 A top-down approach to quality driven architectural engineering – 5th method

K. Lee presented a top-down approach [18] that guides architects to explore and analyze architectural decisions in a top-down manner to effectively handle the complexity of architectural decision space. In
this method, an abstract quality requirement is refined and decomposed into a set of concrete quality requirements that focus on specific aspects of the abstract requirement.

There are two phases in this method. In the first phase, architectural decisions that have global impacts on many aspects are explored and analyzed, i.e. the first phase deals with formulating quality goal requirements, mapping each requirement to its realization of strategic decisions that have global impacts on many quality factors, analyzing risks of the decisions derived from multiple quality goal requirements. In the second phase, architectural decisions that have local impacts on specific aspects are analyzed. Quality goal requirements are refined and tactical decisions that address specific quality factors are analyzed.

5. State of Practice

Case studies can be found at SEI Software Architecture Technology User Network (SATURN) workshop. The web page is http://www.sei.cmu.edu/architecture/saturn. It is a workshop where software system engineers, architects, technical managers and researchers exchange best practices and lessons learned in applying SEI software architecture methods and technologies.

The proceedings of the first and second Software Architecture Technology User Network (SATURN) workshop provide some information about some industrial companies and technology companies that have applied SEI’s methods and technologies in analyzing software quality attributes, such as ABB and Simens.

6. Research Community

6.1 Key conferences

6.1.1 Architecture focus

The following lists some of the conferences that have focus on software architecture.

WICSA (Working IEEE/IFIP Conference on Software Architecture) The purpose of the conference is to software engineering practitioners and researchers from industry and academia to exchange experiences, results and ideas related to all aspects of software architecture, i.e. architectural issues in software design, development and maintenance.

QoSA (International Conference on Quality of Software Architectures) deals with software architecture in general and simultaneously focus on its quality characteristics by addressing the problems of designing good quality software architectures; defining, measuring, evaluating architecture quality and managing architecture quality, tying it upstream to requirements and downstream to implementation; and preserving architecture quality throughout the life time of the system.

QSIC (International Conference on Quality Software) provides a forum to bring together researchers and practitioners working towards improving the quality of software to present new results and exchange ideas in this challenging area.

6.1.2 Software maintenance focus

The following lists some of the conferences that have focus on software maintenance.
ICSM (International Conference on Software Maintenance) provides a forum for software maintenance researchers and practitioners to examine the key issues in software maintenance.

SPLC (Software Product Line Conference) is a leading forum for researchers, practitioners, and educators in the field of software product line to exchange, share and learn technologies and industrial experiences.

CSMR (European Conference on Software Maintenance and Reengineering) promotes discussion and interaction among researchers and practitioners about the development of maintainable systems, and the evolution, migration and reengineering of existing ones.

SOQUA (International Workshop on Software Quality Assurance) brings together researchers, engineers, and practitioners to discuss and evaluate latest challenges, breakthroughs and experiences in the field of software quality assurance, and to identify open issues and future trends in this area.

6.1.3 Software engineering focus
The following lists some of the conferences that have focus on software engineering in general.

Euromicro Conference on Software Engineering and Advanced Applications (SEAA) The aim of the conference is to reflect and represent the continuous changes in technology and application areas, with focus on innovative and advanced applications of software engineering.

ICSE (International Conference on Software Engineering) is the premier software engineering conference, providing a forum for researchers, practitioners and educators to present and discuss the most recent innovations, trends, experiences and concerns in the field of software engineering.

COMPSAC (International Computer Software and Applications Conference) is a major international forum for researchers, practitioners, managers and policy makers interested in computer software and applications. It is one of the major forums for academia, industry and government to discuss the state of the art, new advances and future trends in software technologies and practices.

SEA (Software Engineering and Applications) provides a forum for substantive discussion on the emerging technologies and methodologies being developed to overcome the challenges that exist in all areas of software engineering.

APSEC (Asia-Pacific Systems Engineering Conference) is the leading conference on software engineering and technology in the Asia Pacific region. The conference has both academic and industrial focus and discusses how system engineering is performed as well as the challenges, experiences and lessons learned from expert practitioners.

ESEC/FSE (the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering) provides a lively and outstanding forum where researchers and practitioners to report and discuss recent research results and trends, as well as their impact on practical application in all areas of software engineering.

FASE (Fundamental Approaches to Software Engineering) is a forum to foster feedback between academia and industry by proposing new solutions and evaluating the effectiveness of those solutions in practical contexts. The focus of the conference has been about the ability to produce software of high quality at low cost which is crucial to technological and social progress. An intrinsic characteristic of software that integrates with real-world processes is the need to evolve in order to adjust to new or changing requirements. Maintaining quality, while embracing change, is one of the main challenges of software engineering.
6.2 Leading research groups and researchers

Software Engineering Institute (SEI) is a well known research group within software engineering. It is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University. They focus on software engineering and related disciplines to ensure the development and operations of systems with predictable and improved cost, schedule and quality. Within engineering working area, the software architecture technology initiative has the most relevant connection with our research topic. They are in the leading position in providing methodologies to help software developers use effective architecture-centric practices and better predict the impact of software architectural decisions on quality attributes such as survivability, security, performance, dependability and maintainability.

Software Quality Institute (SQI) at Griffith University provides a focus in Queensland for expertise in software quality and to serve as a catalyst for innovations in software quality techniques. It is engaged in a program of action research with the local software industry and in basic research focused on rigorous computer-assisted program development. The Institute provides consulting and professional support to industry on setting up and managing software quality systems and on using national and international software standards. SQI affiliations with SEI, Standards Australia and other local and international organizations provide industry access to international best practices.

Nokia is one of the companies in the leading positions within software architecture. Nokia Research Center acts as a link between industry research and product development, as well as responding to the product development needs of Nokia’s business groups and carrying out Nokia’s long term research. There are five laboratories at Nokia Research Center, focusing on computing architectures, software and application technologies, technologies within multimedia, networking and radio. The laboratory focusing on computing architecture is the most relevant group. The work of this laboratory is to combine new computing architectures with advanced design methods to master higher levels of system complexity and to increase productivity.

Philips is also one of the companies in the leading positions within software architecture. The healthcare systems architecture research group is one example showing that the company’s focus on challenges in software and architecture. One of the challenges they address is the issue of evolvability of medical system architectures: easier evolution while maintaining throughput, reliability, safety and ease of use. Among others, they also address the issue on how to significantly decrease the lead-time of clinical application development, where they explore the model driven development in combination with the use of Agile methods. Another challenge they are dealing with is to achieve large scale reuse across multiple business groups and they are exploiting the know-how of architecting for variability across multiple views and multiple organization. They investigate as well making optimal use of ‘on-board’ computing resources in order to develop optimal performance systems.

Nenad Medvidovic at University of Southern California is a leading researcher in the area of architecture-based software development. His work focuses on software architecture modeling and analysis; middleware facilities for architectural implementation; product-line architectures; architectural styles; and architecture-level support for software development in highly distributed, mobile, resource constrained, and embedded computing environments.

Christine Hofmeister at Lehigh University is a leading researcher in the area of software architecture and component-based systems.

Vaclav Rajlich at Wayne State University is one of the leading researchers within maintainability. His research includes software engineering in general, methods and tools of software development and program comprehension.
Mira Kajko Mattsson at IT University in Sweden has her research focus within software maintenance. She created two process models of problem management within corrective maintenance CM3: Back-End Problem Management and CM3: Front-End Problem Management.

Geoff Dromey is the director for Software Quality Institute. His research area include programming methodology; formal aspects of program specification and derivation; programming language design; software engineering; software design; software quality; use of computers in education; component technology; software verification. He is famous for his Dromey’s quality model.

7. Research questions

7.1 Definitions

Software architecture of a program or computing system is the structure or structure of the system, which comprise software components, the externally visible properties of those components, and the relationships among them [20].

7.2 Concrete research questions, hypothesis and results

The overall research question is how to maintain software quality during its life cycle. The software quality is targeted to life cycle quality attributes such as maintainability, portability and evolvability. The software systems are existing industrial systems. This overall question can be decomposed into more detailed questions as follows.

7.2.1 Research Question 1

What are the focus quality attribute, concerned characteristics and factors that have impact on the quality attribute?

What is the focus quality attribute? What are the relationships between quality attributes? What are the characteristics that contribute to the quality attribute? What characteristics are of most interest to industry? How to analyze the quality attributes?

Hypothesis

Evolvability will be the focus quality attribute. Of the four categories of maintenance activities as stated in section 1, the corrective maintenance is not the major concern if it is not directly correlated with architecture change.

Research Method

A good understanding of various quality models regarding the quality attributes, their characteristics and the relationships between quality attributes and characteristics is needed in order to be aware of the important factors and characteristics that contribute to software quality and to avoid oversimplification of research problem. This can be achieved through literature survey. Besides, it is also essential to obtain inputs from software architects within industrial software-intensive systems regarding factors that may have impact on software quality (e.g. maintainability). This information can be obtained through interviews.

Research Result

The research result includes a clarification of the focus quality attribute and its characteristics, including characteristics that are derived from industry.
7.2.2 Research Question 2
How to analyze software evolvability?

What are the techniques that can provide feedback on the evolution of quality? What analysis methods are applicable? How they are used to analyze the quality (the process)?

Hypothesis
- Quality models and metrics can be used as criteria and reference for quality measurement.
- Architecture analysis methods can be used to support quality analysis.

Research Method
The information is achieved through literature survey.

Research Result
The research result includes a clarification of the applicable analysis methods and how they are used to analyze the quality attribute as well as the process.

7.2.3 Research Question 3
How to improve software evolvability?

How to model the system? What analysis approaches can be systematically applied on software architecture level to support quality analysis?

Hypothesis
- An analytic model is to be formulated based on the output from previous questions.
- Architecture analysis methods combining with the analytic model and re-factoring can be a solution. (An iterative process maybe)
- A suitable approach should provide assistance in understanding and dealing with trade offs between life cycle properties (maintainability) and other important system quality.
- The approach should include a systematic process that guides the transformation of architecture design.

Research Method
The information is achieved through literature survey and a case study is needed as well.

Research Result
The research result includes a framework for modelling the quality attribute of the software system. The framework will work together with an architecture analysis method in an iterative process, which will guide the architecture transformation through re-factoring in order to obtain better quality attribute.

7.2.4 Research Question 4
How to verify that the system can still achieve satisfactory quality?

Hypothesis
- Scenario-driven analysis methods can be used to analyze if the system can still fulfil related scenarios.

Research Method
Case study is needed.
Research Result
The research result includes a case study of the proposed framework and its combination with an architecture analysis method. The validation is done through the case study.

8. Activities and milestones

8.1 Current activities
There has been mostly course work so far, including
- Research Methodology (finished)
- Progress – Techniques and Technologies (finished)
- Legacy Issues in Industrial Software Systems (finished)
  The course ended in the beginning of April with an internal workshop, where I presented ‘Quality Impact Analysis in Re-factoring’.
- Advanced Component-Based Software Engineering (ongoing)
  The course will end in the beginning of April. A paper on ‘Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles’ was written and submitted to Euromicro CBSE track.
- Research Planning (ongoing)
- Formal Languages, Automater and Theory of Computing (ongoing)

8.2 Planned publications
The planned publications are related to the research questions defined earlier. The investigation results from each research question can be inputs to publications.

8.2.1 Article 1
This article will address quality models, evolvability and its characteristics, including the ones that are of interest to industry. How to analyze evolvability? What analysis methods are applicable and how are they used? How is the process? All these and possibly a case study will be covered in the article.

8.2.2 Article 2
This article will address the framework for modelling and analyzing evolvability of the software system. The combination of the framework and architecture analysis method will be discussed in the paper. Case study is necessary.

8.2.3 Article 3
This article will be based on the second article and further explore the proposed framework. Case study is needed.
8.3 Field and conference trips

8.3.1 Field trip
SAVE-IT is organizing a field trip to Grenoble in the beginning of June 2007. The program has not yet been finalized. The tentative schedule is 3-4 visits combined with some SAVE-IT internal activities. The current list of candidate centres to visit includes:

- Verimag (http://www-verimag.imag.fr/)
- Xerox Research (http://www.xrce.xerox.com/)
- LSR IMAG (http://www-lsr.imag.fr/)
- TIMA (http://tima.imag.fr/)
- INRIA (http://www.inrialpes.fr/jsp/fiche_pagelibre.jsp?CODE=36392593&LANGUE=1)

8.3.2 Conference trips
There will be conference trips when papers are accepted.

8.4 Time plan and milestones

Summer 2007
First article will be hopefully accepted.

December 2007
Second paper is planned for publication.

2008
One or two papers are planned for publication.
Licentiate thesis
9. References