# A Survey of Software Architecture Evolvability

Hongyu Pei Breivold[1], Ivica Crnkovic[2]
[1]*ABB Corporate Research, Industrial Software Systems, 721 78 Västerås, Sweden*
*hongyu.pei-breivold@se.abb.com*
[2]*Mälardalen University, 721 23 Västerås, Sweden*
*ivica.crnkovic@mdh.se*

## Abstract

*For long-lived systems, there is a need to address evolvability (i.e. a system's ability to easily accommodate changes) explicitly during the entire lifecycle. In this report, we undertake a systematic review to obtain an overview of the existing studies in promoting software evolvability at architectural level. The search strategy identified 3036 studies, of which 54 were catalogued as primary studies for this review after using multi-step selection process. The studies are classified into five main categories of themes, including techniques that support quality considerations during software architecture design, architectural quality evaluation, economic valuation, architectural knowledge management and modeling techniques. Four dimensions of factors are identified that exert influence on software evolvability. To cope with these diverse influencing factors, combination of appropriate techniques becomes necessary.*

## 1. Introduction

For long-lived industrial software, the largest part of lifecycle costs is concerned with the evolution of software to meet changing requirements [12]. There is a need to change software on a constant basis with major enhancements within a short timescale in order to keep up with new business opportunities. This puts critical demands on the software system's capability of rapid modification and enhancement to achieve cost-effective software evolution. In this context, software evolvability has appeared as an attribute that describes the software system's capability to accommodate changes [27].

The ever-changing world makes evolvability a strong quality requirement for the majority of software architectures [16]. We have seen at ABB examples of different 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 market situations. Therefore, for such long-lived systems, there is a need to address evolvability explicitly during the entire lifecycle, carry out software evolution efficiently and reliably, and prolong the productive lifetime of the software systems. As software architecture holds a key to the possibility to implement changes in an efficient manner [6], the answer to smooth software evolution is related to the structure of the system. Accordingly, software architecture evolution, architecture evolvability analysis and improvement become a critical part of the software lifecycle.

Software evolvability is a fundamental element for increasing strategic decisions, characteristics, and economic value of the software [20, 76]. The need for greater system evolvability has been recognized [62]. We have also observed this need from various cases in industrial context [19, 22]. Seeing that a system without an adaptable architecture will degenerate sooner than a system based on an architecture that takes changes into account [32], evolvability was identified in these cases as a very important quality attribute that must be maintained.

The notion of evolvability is used in many different ways in the context of software engineering, often with no precise defintion. In addtion, there are many other closely-related quality attributes such as flexibility, maintainability, adaptability and modifiability. For instance, according to [25], adaptability is '*the ability of a software system to accommodate changes in its environment*'. This definition is in line with adaptability definition in ISO 9126 standard [37], in which adaptability is categorized as a subcharacteristic portability and is defined as '*the capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered*'. The scope of the types of changes in these two definitions is rather limited.

In [70], change in the runtime environment is made explicit, and adaptability of a software system or software architecture is defined as '*a qualitative property of its maintainability, and an ability of its components to adapt their functionality, even at runtime, to behavioral and structural changes that* *occur either internally or externally in their operating environment and the requirements of stakeholders' objectives'*.

A broader view of adaptability is given in [51], in which the authors contend that adaptability becomes meaningful when the changes that the software architecture needs to adapt to are specified. Accordingly, software architecture adaptability is defined as '*the degree to which software architecture is adaptable to the change requirement in stakeholders' objectives measured in terms of impact on software architecture elements. Adapatability = <Context, Stakeholder, Change Requirement, Software Architecture Actions>*'. Based on this definition, adaptability is related to the goals of stakeholders and software architecture elements (components and connectors) make reactions to satisfy a certain change requirement in a specified context.

One indicator of quality software is software flexibility as it allows a product to be modified rapidly and cost-effectively for new needs [75]. Flexibility is '*the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed*' [38]. Modifiability is the ability to make changes to a system quickly and cost-effectively [28]. The potential modifications are classified into four categories [40]: (i) extension of capabilities; (ii) deletion of unwanted capabilities; (iii) adaptation to new operating environments; and (iv) restructuring. *Software Evolvability* is defined in [63] as *'Software evolvability is 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*'.

As we can see, this broad nature makes evolvability challenging in practice and research. We have therefore taken into consideration the synonym terms in the search process of the systematic review.

Although there are many research studies for analyzing and achieving software evolvability, they focus on single technique or practice. Besides, no systematic review of software architecture evolvability research has been published previously. Our research is concerned with obtaining a holistic view of analyzing and achieving software evolvability at architectural level as a whole. Therefore, the objectives of this review are to answer the following research questions through systematic literature surveys in current practice and scientific literature:

(i) What results have been reported in the scientific literature regarding the analysis of software evolvability at the architectural level?

(ii) What results have been reported in the scientific literature regarding the factors that affect software evolvability at the architectural level?

This report is organized as follows: Section 2 describes the method used for this review. Section 3 presents the results of the review in five main categories of themes. Section 4 discusses the findings of the review. Section 5 concludes the paper.

## 2. Method

This study is undertaken as a systematic literature review based on the original guidelines as proposed by Kitchenham [43]. The study includes several stages, i.e. the development of a review protocol, the identification of inclusion and exclusion criteria, the search process for relevant publications, quality assessment, data extraction and synthesis.

### 2.1 Review Protocol

We developed a review protocol based on the guidelines and procedures as proposed in [43]. This protocol specifies the background for the survey, research questions, search strategy, study selection criteria, data extraction and synthesis of the extracted data. The protocol was reviewed for comments on the research questions, data sources and search strategy.

### 2.2 Inclusion and Exclusion Criteria

We only consider full papers from peer-reviewed journals, conferences and workshops. The review includes qualitative and quantitative research studies that were published until 2008. Only studies written in English are included. We excluded studies that do not relate to software engineering/development, software architecture and software quality analysis. We also exclude articles in the controversial corner of journals and editorials, prefaces, position papers, summaries of tutorials, workshops, panels and poster sessions. Furthermore, when several duplicated articles of a study exist in different versions that appear as journal papers, conferences and workshop papers, we include only the most complete version of the study and exclude the others.

### 2.3 Search Process

We concentrate on search in peer-reviewed research databases rather than in specific sub-journals. After an initial search of databases, we did an additional reference scanning and analysis in order to find out if we have missed anything, thus to guarantee a representative set of studies. The searched electronic databases include ACM Digital Library, Compendex, IEEE Xplore, ScienceDirect – Elsevier, SpringerLink and ISI Web of Science. The searched results were also mapped towards a core set of studies within software evolution, software architecture and software quality analysis in order to achieve confidence in the completeness of search.

The following search terms are used to find relevant papers:

S1: software architecture AND evolvability
S2: software architecture AND maintainability
S3: software architecture AND extensibility
S4: software architecture AND adaptability
S5: software architecture AND flexibility
S6: software architecture AND changeability
S7: software architecture AND modifiability

All these search terms were combined by using the Boolean OR operator. Thus, an article is included as long as it fulfills one of the search terms. The study selection process was performed through several steps:

1) Search databases and conference proceedings to identify relevant studies;
2) Exclude studies based on the formal exclusion criteria and obvious off-topic contents;
3) Exclude studies based on the titles and abstracts;
4) Obtain primary studies based on full text.

The search strategy identified a total of 3036 articles that were entered into EndNote, which was also used for the subsequent steps for reference storage and sorting. These references were subjected to detailed exclusion criteria and resulted in 731 remaining articles, which were further filtered out by reading titles and abstracts. Studies were excluded if their focus was not software quality analysis or improvement at architectural level. 306 articles were left after this step for full text screening, and detailed quality assessment as described in the next section. In the end, a final figure of 54 articles was identified as primary studies.

## 2.4 Quality Assessment

To guide the interpretation of findings and determine the strength of inferences, we have identified the following quality criteria for appraising the identified studies based on the quality assessment form in [72]:

1) The study is based on research instead of a lessons-learned report or expert opinion;
2) The study's focus is on software architecture and software development quality attribute;
3) The study has an adequate description of the context in which the research was carried out;
4) The research design is appropriate to address the aims of the study;
5) Appropriate data collection methods are used and described;
6) The data analysis is rigorous;
7) The study is of value for research or practice.

Of these quality criteria, the first two were used as the basis for including or excluding a study.

## 2.5 Data Extraction

The data extracted from each study include the source and full reference, main topic area, objectives and aims of the study, statement of research hypothesis if any, type of research (case study, survey, experiment), research method descriptions, definition of software evolvability or its synonyms, data collection, data analysis (e.g. qualitative, quantitative), findings and conclusions.

## 3. Results

Many architectural methods and techniques exist to help analyze and achieve software architecture evolvability. The identified studies are classified into five main groups: techniques that support quality considerations during software architecture design, architectural quality evaluation, economic valuation, architectural knowledge management and modeling techniques.

### 3.1 Quality Considerations Support during Software Architecture Design

Adaptability Evaluation Method (AEM) [70] is an integral part of the Quality-driven Architecture Design and quality Analysis (QADA) methodology [53] specializing in the adaptability aspect. It provides support in requirement engineering, architecture modeling and adaptability evaluation in the architectural models to ensure that the adaptability requirements are met before system implementation. AEM captures the adaptability requirements of the software architecture that will be subsequently considered in the architecture design, provides guidelines on how to model adaptability in the architectural models, and analyzing the candidate architectures to validate whether the adaptability requirements are met.

Non-Functional Requirement (NFR) framework [25] considers adaptability as a key non-functional requirement for evolving systems to ensure adaptability during the process of software development. The NFR framework helps to systematically consider the conflicts and synergies between the NFRs to develop an adaptable architecture through the following five iterative steps: (i) develop the NFR goals and their composition; (ii) develop architectural alternatives; (iii) develop design tradeoffs and rationale; (iv) develop goal criticalities; and (v) architecture evaluation and selection. Another concrete application example of using NFR framework is presented in [24], which describes an NFR approach in developing software system through using design patterns as potential adaptability enhancers.

A systemized method for software architecture analysis throughout the processes of software design and development is described in [21]. Architectural quality goals are mapped into scenarios that measure the goals, mechanisms that realize the scenarios, and analytic models that measure the results. This mapping ensures that design decisions and their rationale are documented so that they can be systematically explored. In addition, as the systems evolve, the analytic models can be used to assess the impact of architectural changes and monitor how architectural evolution over a system's lifetime affects its capability to support predicted modifications.

ArchDesigner [69] is a quality-driven design approach that facilitates the architectural design process. It attempts to best satisfy conflicting stakeholders' quality goals, architectural concerns and project constraints through using optimization techniques. This approach regards the software architecture design problem as a global optimization problem in the sense that the dependencies among different design decisions are maintained, and the selection of any design alternative must not violate global constraints, e.g. stated project constraints.

Global analysis [36] provides a systematic way to identify, accommodate, and describe architecturally significant factors including quality attributes early into the design phase. The influencing factors are classified into three categories, i.e. organizational factors that constrain the design choices, technological and product factors that influence the architecture. Global analysis activities help to uncover the most influential factors, develop strategies for designing the architecture in order to accommodate these factors and reflect future concerns. The global analysis activities continue throughout the architecture design and the results from the activities are used in the central design tasks of each architecture view.

Bosch [17] explicitly considers quality attributes during the design process. The design method he proposed examines three key phases, i.e. functionality-based architecture design, architecture assessment and architecture transformation.

Quality Attribute Workshop (QAW) [5] is a method that engages system stakeholders early in the life cycle to discover a software system's driving quality attribute requirements. The identified quality attribute requirements are elicited in the form of scenarios from the perspectives of diverse groups of stakeholders, covering stimulus, source of stimulus, artifact, environment, response and response measure. The scenarios are classified into use case scenarios, growth scenarios that represent anticipated future changes and exploratory scenarios that stress the system and expose the limits of the current design. The QAW focuses on the involvement of stakeholders and occurs before the creation of the software architecture. In this way, the development team can understand and address the right problem through basing the scenarios on the business goals, and use this information to design the architecture.

Attribute-Driven Design [6] is a recursive method that helps the architect base the design process on the desired quality attribute. Required input to ADD includes known functional requirements, quality attribute requirements, and constraints.

Active Reviews for Intermediate Designs (ARID) [28] is a scenario-based assessment method for evaluating intermediate design or parts of an architecture. It is used to judge if the design of part of the architecture is appropriate for its intended purpose before the development of the complete architecture.

## 3.2 Quality Evaluation at the Software Architecture Level

The foundation for any software system is its architecture, which allows or precludes nearly all of the quality attributes of the system [28]. Accordingly, several architecture evaluation methods have emerged for various purposes, e.g. to compare and identify the strengths and weaknesses in different architecture alternatives, to identify any architectural drift and erosion. From an evolutionary perspective, architecture evaluation is a preventive activity to delay the architectural decay and to limit the effect of software aging [71]. Several studies address how software architecture can be evaluated at the software architecture level with respect to evolvability and its synonyms. We characterize these studies into three groups: experience-based, scenario-based and metric-based evaluation methods.

### 3.2.1 Experience-Based

Experience-based architecture evaluation means that the evaluations are based on the previous experiences and domain knowledge of developers or consultants [1].

Attribute-Based Architectural Style (ABAS) [44] builds on architectural styles by explicitly associating with reasoning frameworks, which are based on quality-attribute-specific models. ABAS consists of four parts: (i) problem description explains the problem being solved by the software structure; (ii) stimuli and response correspond to the condition affecting the system and measurement of the activity as a result of the stimuli; (iii) architectural styles are descriptions of patterns of component interaction; and (iv) analysis constitutes a quality-attribute-specific model that provides a method for reasoning about the behavior of interacting components in the pattern. Examples of these quality-attribute-specific models are modifiability model, reliability model and performance model.

Empirically-Based Architecture Evaluation (EBAE) [50] defines a process for defining and using a number of architectural metrics to evaluate and compare different versions of architectures in terms of maintainability. The main steps include (i) select a perspective for the evaluation; (ii) define and select metrics; (iii) collect metrics; and (iv) evaluate and compare the architectures.

A subset of ALMA [10] is related to the software architecture comparison for optimal candidate architecture. Accordingly, an approach that focuses on quantitatively measuring the stakeholders' views of the benefits and liabilities of software architecture candidates is described in [65]. The data collection in this approach is based on the knowledge, experiences and opinions of the stakeholders. Any disagreements between the participating stakeholders are pinpointed for further investigation.

A knowledge-based approach for assessing evolvability is presented in [30]. The main reasons for knowledge-based approach are the lack of formal and complete architecture documentation, wide scope of assessment, large number of stakholders and distributed development teams. The outcome of the assessment includes the current architecture overview, the main issues found and optionally recommendations for their resolutions. This approach can be used for evaluating the evolutionary path of the software architecture during its lifecycle.

### 3.2.2 Scenario-Based

Scenario-based architecture evaluation means that quality attributes are evaluated by creating scenario profiles that force a concrete description of a quality requirement [54].

Software Architecture Analysis Method (SAAM) [41] is originally created for evaluating modifiability of software architecture although it has been used for other set of quality attributes as well, such as portability and extensibility. The main outputs from a SAAM evaluation include a mapping between the architecture and the scenarios that represent possible future changes to the system, providing indications of potential future complexity parts in the software and estimated amount of work related to the changes.

Architecture Trade-off Analysis Method (ATAM) [28] is a method for evaluating software architectures in terms of quality attribute requirements. It is used to expose the risks, non-risks, sensitivity points and trade-off points in the software architecture. It aims at different quality attributes and supports evaluation of new types of quality attributes. An extension to ATAM is Holistic Product Line Architecture Assessment (HoPLAA) method [57] for the task of assessing product line architectures. It identifies risks at the core architecture level and the indivisual product architecture level. The notion of evolvability points is used to designate a sensistivity point or a tradeoff point that contains at least one variation point. The identification of evolvability point ensures that quality attributes at individual product architecture level do not preclude core architecture quality attributes.

Architecture Level Modifiability Analysis (ALMA) [11] is founded on SAAM [41] and it is a method for analyzing modifiability based on change scenarios that are used to capture future events the system needs to adapt to in its lifecycle. It consists of five steps: (i) set the analysis goal; (ii) describe the software architecture; (iii) elicit change scenarios; (iv) evaluate change scenarios; and (v) interpret the results. The outputs from an ALMA evaluation include: (i) maintenance prediction to estimate the required effort for system modification to accommodate future changes; (ii) risk assessment to identify the types of changes that the system shows inability to adapt to; and (iii) software architecture comparison for optimal candidate architecture. A subset of ALMA is related to risk assessment which focuses on exposing the boundaries of software architecture with respect to flexibility using complex scenarios [48].

Scenario-Based Architecture Reengineering (SBAR) [9] considers multiple quality attributes and classifies them into development-oriented and operational related. Another scenario-based software architecture assessment method based on SAAM, ATAM and SBAR is presented in [29]. It is used for evaluating evolvability of software product family architecture towards forthcoming requirements. The output includes the potential flaws and evolutionary path of the software.

### 3.2.3 Metric-Based

A process-oriented metric for software architecture adaptability is described in [26], which analyzed the degree of adaptability through intuitive decomposition of goals and intitive scoring for the goal satifying level of software architecture. As the method depends much on the intuition and expert expertise, a quantitative metric-based aproach that evaluates software architecture adaptability is proposed in [51]. This approach supports decision making for choosing architecture candidates that meet the stakeholders' adaptability goals. The adaptability goals are expressed in terms of adaptability scenario profiles. The impact of each scenario profile is measured through two metrics, i.e. IOSA (impact on the software architecture) and ADSA (adaptability degree of software architecture).

A quality model provides a framework for quality assessment. It aims at describing complex quality criteria through breaking them down into concrete subcharacteristics that are measured using metrics. 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 [55], Dromey's quality model [31], Boehm's quality model [15], ISO 9126 [37] and FURPS quality model [34].

Several metrics have been proposed for evaluating evolvability. Ramil and Lehman proposed metrics based on implementation change logs [60] and computation of metrics using the number of modules in a software system [49]. Another set of metrics is based on software life span and software size [67]. In [56], 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 [23].

## 3.3 Economic Valuation in Determining the Level of Uncertainty

Cost Benefit Analysis Method (CBAM) [39] builds upon ATAM [42] and is an architecture-centric economic modeling approach that helps to address the long-term benefits with regards to a change and its complete product lifecycle implications. It is an approach for deciding how to prioritize changes to an architecture based on perceived difficulty and utility. This method quantifies design decisions in terms of cost and benefit analysis to determine the level of uncertainty. The uncertainties encountered in the software architecture evolution come from understanding how architectural decisions map onto quality attribute responses; how architectural decisions map onto costs; and how quality attribute responses map onto benefits. These uncertainties are elicited and recorded through the six steps of the CBAM: (i) choosing scenarios and architectural strategies; (ii) assessing quality attribute benefits; (iii) quantifying the architectural strategies' benefits; (iv) quantifying the architectural strategies' costs and schedule implications; (v) calculate desirability; and (vi) make decisions. A related economics-driven method is Architecture Improvement Workshop (AIW)[1] which also builds upon ATAM and values architectural decisions in relation to quality attributes.

Software architecture decisions carry economic value in the form of real options [4, 64]. Options offer flexibility and consider the architectural evolution over time [2]. An approach that considers cost, value and alignment with business goals to support architectural evolution is described in [58]. The approach guides the selection of design patterns, the elicitation of architecturally significant requirements, and the valuation of an architecture in terms of design decisions with multiple quality attributes viewpoints. Another application of real options theory is described in [3], which provides insights into architectural flexibility and investment decisions related to the evolution of software systems. To cope with uncertainty and mitigate risks in the investment, a set of probable changes is examined, as well as the added value of the embedded flexibility in response to these changes. According to [3], the added value is strategic in essence. Some examples are (i) accumulated savings through enduring the change without violating architectural integrity; (ii) supporting future growth; (iii) capability of responding to competitive forces and changing market conditions.

Another way to address economic valuation is through predicting the required effort for a maintenance task. A subset of ALMA [11] is related to maintenance cost prediction, i.e. architecture level maintenance prediction [8] uses change scenarios that represent perfective and adaptive maintenance tasks to concretize the maintainability requirements in the life cycle of the system, evaluates the architecture using sceanrio scripting, and calculate the expected effort for each change scenario based on the analysis of how the change could be implemented and the amount of required changed code.

A study in [59] proposes a model-based approach to strategically determine an appropriate degree of architectural flexibility through four strategic elements, i.e. feature prioritization, schedule range estimation, core capability determination and architecture flexibility determination given particular schedule

[1]http://www.sei.cmu.edu/architecture/products_services/aiw.html

constraints. In this way, the risk of violating schedule, cost and quality constraints is lowered.

## 3.4 Architectural Knowledge Management

Architectural knowledge consists of architecture design, design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is. An explicit representation of architectural knowledge is helpful for evolving quality systems and assessing future evolutionary capabilities of a system [45].

Apart from using change scenarios and change cases to explicitly model variability and describe the future evolutionary capabilities as most software architecture analysis methods do, the authors in [46] believe that it is also useful to explicitly model invariability, i.e. things that are assumed will not change. This information can be used for the explicit modeling of assumptions and provides additional what-if scenarios for software architecture assessment, i.e. what if a certain assumption proves to be invalid. Assumptions are design decisions and rationale that are made out of personal experience and background, domain knowledge, budget constraints, available expertise etc. The assumptions are classified into technical assumptions that concern the technical environment a system is running in, organizational assumptions that concern the organizational aspects in a company, and managerial assumptions that reflect the decisions taken to achieve business objectives. Explicit representation of these assumptions provides traceability between the software architecture evolution and the early-made assumptions, and augments design decisions in the face of uncertainties when predicting the future user requirement changes. In order to better assess the future evolutionary capabilities of a system, Recovering Architectural Assumptions Method (RAAM) [61] was developed to make the assumptions explicit. This method is appropriate when historical information of software system evolution is available and when there is access to the development team for interviews.

Design erosion is inevitable due to the ever-changing requirements [74]. To assess architectural erosion and track software evolution, an architecture assessment model is described in [14], objectively measuring the extent of deviation in terms of functional and structural divergence. In addition, the loss of system functionality and architectural structure as a software system evolves is represented through functional and structural erosions as erosion indicators.

As design rationale captures the knowledge and grounds that shape a software architecture, documenting design rationale is another approach that is used to maintain and evolve architectural artifacts [68] in order to allow unanticipated changes in the software without compromising software integrity and to evolve in a controlled way [13].

The study in [33] generalizes architectural styles, patterns and similar concepts by introducing the concept of architectural constraints. It is argued that architectural constraints strongly influence the quality of architectural design process and the improvement of software quality.

## 3.5 Modeling Techniques

One source of requirement changes is related to the change in business rules. Evolvable software architecture considers business rule as a key component due to its high impact on software and business process. A study in [73] proposes therefore a Business Rule Model to capture and specify business rules and a Link Model to relate these business rules to the metamodel level of software design elements. The explicit consideration and modeling of business rules facilitate the improvement of software evolvability.

Another way to achieve or improve evolvability is through considering the relations between requirements, architectural elements and implementation. A model-based approach for modeling the traceability links is presented in [18], in which the indicators for problem situation are formally defined as well as the corrsponding actions for problem resolution.

A quality-driven software reengineering model [66] is proposed to address the evolution of system requirements and software architecture. This approach adopts NFR Framework [23] and the concept of soft goals to support the systematic modeling of the design rationale through a soft-goal interdependency graph.

A framework for modeling various types of relevant information and a set of architectural views for reengineering, analyzing, and comparing software architectures is presented in [52]. This approach builds upon a scenario-based approach and captures and assesses software architectures for evolution and reuse. The types of information for modeling include: (i) *Stakeholder information* describes stakeholders' objectives, which provide boundaries for analysis; (ii) *Architecture information* refers to design principles or architectural objectives; (iii) *Quality information* refers to non-functional attributes; (iv) *Scenarios* describe the use cases of the system to capture the system's functionality. Scenarios that are not directly supported by the current system can be used to detect possible flaws or to assess the architecture's support for potential enhancements. Scenarios are derived from the stakeholder and architectural objectives, as well as desired system quality attributes.

## 4. Discussions

Based on the review, four dimensions of factors that exert influence on software architecture evolvability are identified, i.e. (i) business, (ii) technology, (iii) development process and (iv) organization. From business perspective, system requirements evolve because stakeholders' needs and expectations change, the context in which the software operate changes [14], and business models evolve [73]. From technology perspective, many unknown, uncontrollable technological and environmental constraints outweigh design principles [35]. Assumptions shaping software architectures [46] consist of technical assumptions that concern the technical environment a system is running in, organizational assumptions that concern the organizational aspects in a company, and managerial assumptions that reflect the decisions taken to achieve business objectives.

Patterns of risk themes that influence evolvability are categorized into architecture, process and organization [7]. From architecture perspective, the lack of attention to potential growth paths and unknown requirements result in the failure to achieve modifiability goals. From process perspective, requirement is identified as one risk theme due to its nature of being uncertain or rapid-changing, e.g. lack of attention to important concerns of key stakeholders, lack of consistent marketing input, and disagreement among stakeholders. From organization perspective, one risk theme is the unrecognized need, arising from the failure to consider architecture aspects in the overall system construction.

To cope with the above diverse influencing factors, a spectrum of techniques and approaches has been identified that promote software evolvability from a specific perspective or architecture-centric activity in the software lifecycle:

- Most of the techniques that support quality considerations during software architecture design help identify key quality attribute requirements early.
- In the subsequent iteration when the architecture starts to take form, architectural quality evaluation methods help elicit and refine additional quality attribute requirements and scenarios.
- Economic valuation methods provide more details on architectural decisions' business consequences and assist development teams in choosing among architectural options.
- Architectural knowledge management and modeling techniques add value by modeling traceability and visualizing corresponding impact of the evolution of software architecture artifacts.

Each of the aforementioned approaches has its strengths and shortcomings, and has its specific context that it is appropriate for. For instance, most scenario-based software architecture analysis methods share the strength of being able to concretize the driving quality attribute requirements, but they also share the weakness of being optimistic in change scenario elicitation due to the unpredictable nature of changes as well as the stakeholders' short horizon in foreseeing future changes [47]. Some architectural knowledge management approaches can complement scenario-based methods and address this weakness through explicit representation of invariabilities to provide additional what-if scenarios. As evolvability needs to be addressed and maintained throughout the complete software lifecycle, combination of appropriate techniques becomes necessary for software systems to cope with the diverse types of influencing factors.

## 5. Conclusions

The systematic review of studies in promoting software evolvability at architectural level identified 3036 studies, of which 54 were catalogued as primary studies. The studies are classified into five main categories of themes, i.e. techniques that support quality considerations during software architecture design, architectural quality evaluation, economic valuation, architectural knowledge management and modeling techniques. Each of these approaches has its strengths and shortcomings, and has its specific context that it is appropriate for. Accordingly, a combination of appropriate techniques is necessary to cope with the diverse influencing factors to evolvability exhibited in four dimensions, i.e. business, technology, organization and process.

## References

[1] Avritzer, A. and Weyuker, E. J., "Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews," *Empirical Software Engineering,* vol. 4, pp. 199-215, 1999.

[2] Bahsoon, R. and Emmerich, W., "ArchOptions: a real options-based model for predicting the stability of software architectures," p. 38.

[3] Bahsoon, R. and Emmerich, W., "Evaluating architectural stability with real options theory," 2004, pp. 443-447.

[4] Baldwin, C. Y. and Clark, K. B., *Design rules: Volume 1: The power of modularity*: Mit Press Cambridge, MA, 2000.

[5] Barbacci, M. R., Ellison, R., Lattanze, A. J., Stafford, J. A., Weinstock, C. B., and Wood, W. G., "Quality attribute workshops (qaws)," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2003.

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

[7] Bass, L., Nord, R., Wood, W., Zubrow, D., and Ozkaya, I., "Analysis of architecture evaluation data," *Journal of Systems and Software,* vol. 81, pp. 1443-1455, 2008.

[8] Bengtsson, P. and Bosch, J., "Architecture level prediction of software maintenance," 1999, pp. 139-147.

[9] Bengtsson, P. and Bosch, J., "Scenario-based software architecture reengineering," 1998, pp. 308-317.

[10] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H., "Architecture-level modifiability analysis (ALMA)," *Journal of Systems and Software,* vol. 69, pp. 129-147, 2004.

[11] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H., "Architecture-level modifiability analysis (ALMA)," *The Journal of Systems & Software,* vol. 69, pp. 129-147, 2004.

[12] Bennett, K., "Software evolution: past, present and future," *Information and Software Technology,* vol. 38, pp. 673-680, 1996.

[13] Bennett, K. H. and Rajlich, V. T., "Software maintenance and evolution: a roadmap," 2000, pp. 73-87.

[14] Bhattacharya, S. and Perry, D. E., "Architecture assessment model for system evolution," Mumbai, India, 2007.

[15] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., MacLeod, G. J., and Merritt, M. J., *Characteristics of software quality*: North-Holland, 1978.

[16] Borne, I., Galal, G. H., Evans, H., and Andrade, L. F., "Object-oriented architectural evolution," in *14th European Conference on Object Oriented Programming (ECOOP 2000)*, Cannes, France, 2000, pp. 138-149.

[17] Bosch, J., *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*: Addison-Wesley, 2000.

[18] Brcina, R. and Riebisch, M., "Architecting for evolvability by means of traceability and features," in *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, 2008, pp. 72-81.

[19] Breivold, H. P., Crnkovic, I., and Eriksson, P. J., "Analyzing Software Evolvability," 2008.

[20] Cai, Y. and Huynh, S., "An evolution model for software modularity assessment," 2007, pp. 3-3.

[21] Carriere, S. J., Kazman, R., and Woods, S. G., "Assessing and maintaining architectural quality," Amsterdam, Neth, 1999, pp. 22-30.

[22] Christian, D. R., "Continuous evolution through software architecture evaluation: a case study," *J. Softw. Maint. Evol.: Res. Pract,* vol. 18, pp. 351-383, 2006.

[23] Chung, L., *Non-Functional Requirements in Software Engineering*: Springer, 2000.

[24] Chung, L., Cooper, K., and Yi, A., "Developing adaptable software architectures using design patterns: an NFR approach," *Computer Standards & Interfaces,* vol. 25, pp. 253-260, 2003.

[25] Chung, L. and Subramanian, N., "Adaptable architecture generation for embedded systems," *Journal of Systems and Software,* vol. 71, pp. 271-295, 2004.

[26] Chung, L., Subramanian, N., and Ieee Computer Society, I. C. S., "Process-oriented metrics for software architecture adaptability," in *5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 2001, pp. 310-311.

[27] Ciraci, S. and Van Den Broek, P., "Evolvability as a Quality Attribute of Software Architectures," 2006, pp. 6-7.

[28] Clements, P., Kazman, R., and Klein, M., *Evaluating Software Architectures: Methods and Case Studies*: Addison-Wesley, 2002.

[29] Del Rosso, C., "Continuous evolution through software architecture evaluation: A case study," *Journal of Software Maintenance and Evolution,* vol. 18, pp. 351-383, 2006.

[30] Del Rosso, C. and Maccari, A., "Assessing the architectonics of large, software-intensive systems using a knowledge-based approach," Mumbai, India, 2007.

[31] Dromey, R. G., "Cornering the Chimera," *IEEE Software,* vol. 13, pp. 33-43, 1996.

[32] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design patterns: elements of reusable object-oriented software*, 1995.

[33] Giesecke, S., Hasselbring, W., and Riebisch, M., "Classifying architectural constraints as a basis for software quality assessment," *Advanced Engineering Informatics,* vol. 21, pp. 169-179, 2007.

[34] Grady, R. B. and Caswell, D. L., *Software metrics: establishing a company-wide program*: Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1987.

[35] Graham, T. C. N., Rick, K., and Chris, W., "Agility and Experimentation: Practical Techniques for Resolving Architectural Tradeoffs," in *Proceedings of the 29th international conference on Software Engineering*: IEEE Computer Society, 2007.

[36] Hofmeister, C., Nord, R., and Soni, D., *Applied software architecture*: Addison-Wesley Professional, 2000.

[37] ISO9126, "ISO/IEC 9126-1, International Standard, Software Engineering. Product Quality – Part 1: Quality Model."

[38] Jay, F. and Mayer, R., "IEEE standard glossary of software engineering terminology," *IEEE Std,* pp. 610.12-1990, 1990.

[39] Kazman, R., Asundi, J., and Klein, M., "Quantifying the costs and benefits of architectural decisions," 2001, pp. 297-306.

[40] Kazman, R., Bass, L., Abowd, G., and Webb, M., "SAAM: a method for analyzing the properties of software architectures," in *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, 1994, pp. 81-90.

[41] Kazman, R., Bass, L., Abowd, G., and Webb, M., "SAAM: A Method for Analyzing the Properties of Software Architectures," *INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING,* vol. 16, pp. 81-81, 1994.

[42] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., and Ieee, I., "The architecture tradeoff analysis method," in *4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 98)*, Monterey, Ca, 1998, pp. 68-78.

[43] Kitchenham, B., "Procedures for performing systematic reviews," *Keele University TR/SE-0401/NICTA Technical Report 0400011T,* vol. 1, 2004.

[44] Klein, M., Kazman, R., Bass, L., Carriere, J., Barbacci, M., and Lipson, H., *Attribute-Based Architecture Styles*: Kluwer, BV Deventer, The Netherlands, The Netherlands, 1999.

[45] Kruchten, P., Lago, P., and van Vliet, H., "Building up and reasoning about architectural knowledge," *LECTURE NOTES IN COMPUTER SCIENCE,* vol. 4214, p. 43, 2006.

[46] Lago, P. and van Vliet, H., "Explicit assumptions enrich architectural models," 2005, p. 206.

[47] Lassing, N., Rijsenbrij, D., and van Vliet, H., "How well can we predict changes at architecture design time?," *Journal of Systems and Software,* vol. 65, pp. 141-153, 2003.

[48] Lassing, N., Rijsenbrij, D., and van Vliet, H., "On software architecture analysis of flexibility, Complexity of changes: Size isn't everything," 1999, pp. 1103-1581.

[49] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M., "Metrics and laws of software evolution - the nineties view," Albuquerque, NM, USA, 1997, pp. 20-32.

[50] Lindvall, M., Tvedt, R. T., and Costa, P., "An Empirically-Based Process for Software Architecture Evaluation," *Empirical Software Engineering,* vol. 8, pp. 83-108, 2003.

[51] Liu, X. and Wang, Q., "Study on application of a quantitative evaluation approach for software architecture adaptability," in *5th International Conference on Quality Software (QSIC 2005),* Melbourne, AUSTRALIA, 2005, pp. 265-272.

[52] Lung, C. H., Bot, S., Kalaichelvan, K., and Kazman, R., "An approach to software architecture analysis for evolution and reusability," 1997.

[53] Matinlassi, M., "Quality-driven software architecture model transformation," 2005.

[54] Mattsson, M., Grahn, H., and Mårtensson, F., "Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability," 2006.

[55] McCall, J. A., Richards, P. K., Walters, G. F., United, S., Electronic Systems, D., Force, A., Rome Air Development, C., and Systems, C., *Factors in Software Quality*: NTIS, 1977.

[56] Nary, S. and Chung, L., "Process-oriented metrics for software architecture evolvability," Helsinki, Finland, 2003, pp. 65-70.

[57] Olumofin, F. G. and Mišic, V. B., "A holistic architecture assessment method for software product lines," *Information and Software Technology,* vol. 49, pp. 309-323, 2007.

[58] Ozkaya, I., Kazman, R., and Klein, M., "Quality-Attribute Based Economic Valuation of Architectural Patterns," 2007, pp. 5-5.

[59] Port, D. and LiGuo, H., "Strategic architectural flexibility," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, 2003, pp. 389-396.

[60] Ramil, J. F. and Lehman, M. M., "Metrics of software evolution as effort predictors - a case study," San Jose, CA, USA, 2000, pp. 163-172.

[61] Roeller, R., Lago, P., and van Vliet, H., "Recovering architectural assumptions," *The Journal of Systems & Software,* vol. 79, pp. 552-573, 2006.

[62] Rowe, D. and Leaney, J., "Evaluating evolvability of computer based systems architectures-an ontological approach," 1997, pp. 24-28.

[63] Rowe, D., Leaney, J., and Lowe, D., "Defining systems evolvability-a taxonomy of change," *Change,* vol. 94, pp. 541-545, 1994.

[64] Sullivan, K. J., Chalasani, P., Jha, S., and Sazawal, V., "Software design as an investment activity: a real options perspective," *Real Options and Business Strategy: Applications to Decision Making,* pp. 215–262, 1999.

[65] Svahnberg, M., "An industrial study on building consensus around software architectures and quality attributes," *Information and Software Technology,* vol. 46, pp. 805-818, 2004.

[66] Tahvildari, L., Kontogiannis, K., and Mylopoulos, J., "Quality-driven software re-engineering," *Journal of Systems and Software,* vol. 66, pp. 225-239, 2003.

[67] Tamai, T. and Torimitsu, Y., "Software lifetime and its evolution process over generations," Orlando, FL, USA, 1992, pp. 63-9.

[68] Tang, A., Babar, M. A., Gorton, I., and Han, J., "A survey of architecture design rationale," *Journal of Systems and Software,* vol. 79, pp. 1792-1804, 2006.

[69] Tariq, A.-N., Ian, G., Muhammed Ali, B., Fethi, R., and Boualem, B., "A quality-driven systematic approach for architecting distributed software applications," in *Proceedings of the 27th*

*international conference on Software engineering* St. Louis, MO, USA: ACM, 2005.

[70] Tarvainen, P., "Adaptability evaluation of software architectures; A case study," in *31st Annual International Computer Software and Applications Conference*, Beijing, PEOPLES R CHINA, 2007, pp. 579-584.

[71] Tonu, S. A., Ashkan, A., and Tahvildari, L., "Evaluating architectural stability using a metric-based approach," 2006.

[72] Tore, D., Torgeir, D., and yr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.,* vol. 50, pp. 833-859, 2008.

[73] Wan-Kadir, W. M. N. and Loucopoulos, P., "Relating evolving business rules to software design," *Journal of Systems Architecture,* vol. 50, pp. 367-382, 2004.

[74] van Gurp, J. and Bosch, J., "Design erosion: problems and causes," *Journal of Systems and Software,* vol. 61, pp. 105-119, 2002.

[75] Wang, E. T. G., Ju, P. H., Jiang, J. J., and Klein, G., "The effects of change control and management review on software flexibility and project performance," *Information & Management,* vol. 45, pp. 438-443, 2008.

[76] Weiderman, N. H., Bergey, J. K., Smith, D. B., and Tilley, S. R., "Approaches to Legacy System Evolution," 1997.

## Appendix: Studies Included in the Review[2]

[S1] Aoyama, M., "Continuous and discontinuous software evolution: aspects of software evolution across multiple product lines," 4th international workshop on Principles of software evolution, 2001, pp. 87-90.

[S2] Babar, M. A. and Gorton, I., "A tool for managing software architecture knowledge," ICSE Workshop Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent 2007.

[S3] Babar, M. A., Gorton, I., and Jeffery, R., "Capturing and using software architecture knowledge for architecture-based software development," QSIC 2005, pp. 169-176.

[S4] Bahsoon, R. and Emmerich, W., "ArchOptions: a real options-based model for predicting the stability of software architectures," 5 th International Workshop on Economic-Driven Software Engineering Research.

[S5] Bahsoon, R. and Emmerich, W., "Evaluating architectural stability with real options theory," Chicago, IL, United states, ICSM 2004, pp. 443-447.

[S6] Bass, L., Clements, P., and Kazman, R., Software architecture in practice: Addison-Wesley Professional, 2003.

[S7] Bengtsson, P. and Bosch, J., "Architecture level prediction of software maintenance," in 3rd European Conference on Software Maintenance and Reengineering (CSMR 99), Amsterdam, Netherlands, 1999, pp. 139-147.

[S8] Bengtsson, P. and Bosch, J., "Scenario-based software architecture reengineering," Internation Conference on Software Reuse 1998, pp. 308-317.

[S9] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H., "Architecture-level modifiability analysis (ALMA)," Journal of Systems and Software, vol. 69, pp. 129-147, 2004.

[S10] Bhattacharya, S. and Perry, D. E., "Architecture assessment model for system evolution," WICSA, Mumbai, India, 2007.

[S11] Bosch, J., Design and use of software architectures: adopting and evolving a product-line approach: Addison-Wesley Professional, 2000.

[S12] Breivold, H.P., I. Crnkovic, and P.J. Eriksson, Analyzing Software Evolvability. COMPSAC'08. 32nd Annual IEEE International Computer Software and Applications, 2008.

[S13] Brcina, R. and Riebisch, M., "Architecting for evolvability by means of traceability and features," in Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on, 2008, pp. 72-81.

[S14] Browning, T. R. and Honour, E. C., "Measuring the life-cycle value of enduring systems," Systems Engineering, vol. 11, 2008.

[S15] Burge, J. E. and Brown, D. C., "Software Engineering Using RATionale," The Journal of Systems & Software, vol. 81, pp. 395-413, 2008.

[S16] Capilla, R., Nava, F., and Dueas, J. C., "Modeling and documenting the evolution of architectural design decisions," ICSE Workshop Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent 2007, pp. 9-9.

[S17] Capilla, R., Nava, F., Pérez, S., and Dueñas, J. C., "A web-based tool for managing architectural design decisions," ACM SIGSOFT software engineering notes, vol. 31, 2006.

[S18] Carriere, S. J., Kazman, R., and Woods, S. G., "Assessing and maintaining architectural quality," Amsterdam, Neth, CSMR 1999, pp. 22-30.

[S19] Chung, L., Cooper, K., and Yi, A., "Developing adaptable software architectures using design patterns: an NFR approach," Computer Standards & Interfaces, vol. 25, pp. 253-260, 2003.

[S20] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J., Non-functional requirements in software engineering: Springer, 1999.

[2] The appendix includes the 54 papers that were identified from the search process (conducted in April, 2009) as described in the report. An additional 4 papers were added when we performed a complementary search in the end of August, 2009 in order to cover the publications within the period of 2008 and the first quarter of 2009. But some studies in the second quarter might not have been indexed in the databases.

[S21] Chung, L., Subramanian, N., and Ieee Computer Society, I. C. S., "Process-oriented metrics for software architecture adaptability," in 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, 2001, pp. 310-311.

[S22] Clements, P., Kazman, R., and Klein, M., Evaluating software architectures: methods and case studies: Addison-Wesley, 2006.

[S23] Del Rosso, C., "Continuous evolution through software architecture evaluation: A case study," Journal of Software Maintenance and Evolution, vol. 18, pp. 351-383, 2006.

[S24] Del Rosso, C. and Maccari, A., "Assessing the architectonics of large, software-intensive systems using a knowledge-based approach," WICSA Mumbai, India, 2007.

[S25] Engel, A. and Browning, T. R., "Designing systems for adaptability by means of architecture options," Systems Engineering, vol. 11, 2008.

[S26] Farenhorst, R., Izaks, R., Lago, P., and van Vliet, H., "A Just-In-Time Architectural Knowledge Sharing Portal," WICSA 2008, pp. 125-134.

[S27] Fricke, E., Gebhard, B., Negele, H., and Igenbergs, E., "Coping with changes: Causes, findings, and strategies," Systems Engineering, vol. 3, pp. 169-179, 2000.

[S28] Fricke, E. and Schulz, A. P., "Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle," Systems Engineering, vol. 8, 2005.

[S29] Giesecke, S., Hasselbring, W., and Riebisch, M., "Classifying architectural constraints as a basis for software quality assessment," Advanced Engineering Informatics, vol. 21, pp. 169-179, 2007.

[S30] Hofmeister, C., Nord, R., and Soni, D., Applied Software Architecture: A Practical Guide for Software Designers: Addison-Wesley Professional, 2000.

[S31] Ipek, O., Rick, K., and Mark, K., "Quality-Attribute Based Economic Valuation of Architectural Patterns," in Proceedings of the First International Workshop on The Economics of Software and Computation: IEEE Computer Society, 2007.

[S32] Jansen, A., Bosch, J., and Avgeriou, P., "Documenting after the fact: Recovering architectural design decisions," The Journal of Systems & Software, vol. 81, pp. 536-557, 2008.

[S33] Jansen, A., Van der Ven, J., Avgeriou, P., and Hammer, D. K., "Tool support for architectural decisions," WICSA 2007, pp. 4-4.

[S34] Kazman, R., Bass, L., Abowd, G., and Webb, M., "SAAM: a method for analyzing the properties of software architectures," in Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on, 1994, pp. 81-90.

[S35] Kazman, R. and Klein, M., "Designing and analyzing software architectures using ABASs," ICSE 2000.

[S36] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., and Ieee, I., "The architecture tradeoff analysis method," in 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 98), Monterey, Ca, 1998, pp. 68-78.

[S37] Lago, P., van Vliet, H., and acm, "Explicit assumptions enrich architectural models," in 27th International Conference on Software Engineering (ICSE 2005), St Louis, MO, 2005, pp. 206-214.

[S38] Lassing, N., Bengtsson, P., van Vliet, H., and Bosch, J., "Experiences with ALMA: Architecture-Level Modifiability Analysis," Journal of Systems and Software, vol. 61, pp. 47-57, 2002.

[S39] Lassing, N., Rijsenbrij, D., and van Vliet, H., "How well can we predict changes at architecture design time?," Journal of Systems and Software, vol. 65, pp. 141-153, 2003.

[S40] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M., "Metrics and laws of software evolution-the nineties view," 4th International Software Metrics Symposium 1997.

[S41] Liu, X. and Wang, Q., "Study on application of a quantitative evaluation approach for software architecture adaptability," in 5th International Conference on Quality Software (QSIC 2005), Melbourne, AUSTRALIA, 2005, pp. 265-272.

[S42] Lung, C. H., Bot, S., Kalaichelvan, K., and Kazman, R., "An approach to software architecture analysis for evolution and reusability," conference of the Centre for Advanced Studies on Collaborative research 1997.

[S43] Olumofin, F. G. and Misic, V. B., "A holistic architecture assessment method for software product lines," Information and Software Technology, vol. 49, pp. 309-323, 2007.

[S44] Port, D. and LiGuo, H., "Strategic architectural flexibility," in Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, 2003, pp. 389-396.

[S45] Ramil, J. F. and Lehman, M. M., "Metrics of software evolution as effort predictors-a case study," ICSM 2000, pp. 163-172.

[S46] Rick, K., Jai, A., and Mark, K., "Quantifying the costs and benefits of architectural decisions," in Proceedings of the 23rd International Conference on Software Engineering Toronto, Ontario, Canada: IEEE Computer Society, 2001.

[S47] Roeller, R., Lago, P., and van Vliet, H., "Recovering architectural assumptions," Journal of Systems and Software, vol. 79, pp. 552-573, 2006.

[S48] Subramanian, N. and Chung, L., "Process-oriented metrics for software architecture evolvability," in 6th International Workshop on Principles of Software Evolution, Helsinki, Finland, 2003, pp. 65-70.

[S49] Sullivan, K. J., Griswold, W. G., Cai, Y., and Hallen, B., "The structure and value of modularity in software design," 8th European software engineering conference held jointly with 9th ACM

SIGSOFT international symposium on Foundations of software engineering 2001, pp. 99-108.

[S50] Svahnberg, M., "An industrial study on building consensus around software architectures and quality attributes," Information and Software Technology, vol. 46, pp. 805-818, 2004.

[S51] Tahvildari, L., Kontogiannis, K., and Mylopoulos, J., "Quality-driven software re-engineering," Journal of Systems and Software, vol. 66, pp. 225-239, 2003.

[S52] Tamai, T. and Torimitsu, Y., "Software lifetime and its evolution process over generations," ICSM 1992, pp. 63-69.

[S53] Tang, A., Avgeriou, P., Jansen, A., Capilla, R., and Ali-Babar, M., "A Comparative Study of Architecture Knowledge Management Tools," Journal of Systems and Software, 2009.

[S54] Tang, A., Babar, M. A., Gorton, I., and Han, J., "A survey of architecture design rationale," Journal of Systems and Software, vol. 79, pp. 1792-1804, 2006.

[S55] Tariq, A.-N., Ian, G., Muhammed Ali, B., Fethi, R., and Boualem, B., "A quality-driven systematic approach for architecting distributed software applications," in Proceedings of the 27th international conference on Software engineering St. Louis, MO, USA: ACM, 2005.

[S56] Tarvainen, P., "Adaptability evaluation of software architectures; A case study," in 31st Annual International Computer Software and Applications Conference, Beijing, PEOPLES R CHINA, 2007, pp. 579-584.

[S57] Wan-Kadir, W. M. N. and Loucopoulos, P., "Relating evolving business rules to software design," Journal of Systems Architecture, vol. 50, pp. 367-382, 2004.

[S58] Zhu, L., Babar, M. A., and Jeffery, R., "Mining patterns to support software architecture evaluation," WICSA 2004, pp. 25-34.