A Systematic Review of Software Evolvability

Hongyu Pei Breivold
ABB Corporate Research

Hongyu.pei-breivold@se.abb.com

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 paper, we undertake a systematic review to obtain an overview of the existing studies in promoting software evolvability at architectural level. The search strategy identified 58 studies that 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. The review investigates what is currently known about software evolvability architecting at architecture level. Implications for research and practice are presented.

Keywords

Systematic review, software architecture, software evolvability.

1. INTRODUCTION

For long-lived industrial software, the largest part of lifecycle costs is concerned with the evolution of software to meet changing requirements [7]. This puts critical demands on 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 *'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'* [44].

The ever-changing world makes evolvability a strong quality requirement for the majority of software architectures [11, 43]. In our previous study at ABB [13], and in other studies [15], we have seen examples of different industrial systems that have a lifetime of 10-30 years and are continuously changing. 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. As software evolvability is a fundamental element for increasing strategic decisions, characteristics, and economic value of the software [14, 50], for such long-lived systems, there is a need to address evolvability explicitly during the entire lifecycle and prolong the productive lifetime of the software systems. Seeing that a system without an adaptable architecture will degenerate sooner than a system based on an architecture that takes changes into account [24], evolvability was identified in these cases as a very important quality attribute that must be continuously maintained during their lifecycle. However, although there are many research studies for analyzing and promoting software evolvability, they focus on a particular technique or practice; No systematic review of software architecture evolvability research has been published previously (to our best knowledge).

The foundation for any software system is its architecture, which allows or precludes nearly all of the quality attributes of the system [18]. Our research is therefore concerned with obtaining a holistic view of the existing studies in analyzing and achieving software evolvability at architectural level through systematic review [28]. The objectives are to answer the following question: (i) *What results have been reported in the scientific literature regarding the analysis and achievement of software evolvability at the architectural level?* (ii) *What are the implications of the studies for research community and software practice?*

The remainder of this paper is structured 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, and implications for research and practice. 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 [28]. The study includes several stages: (i) development of a review protocol; (ii) identification of inclusion and exclusion criteria; (iii) the search process for relevant publications; (iv) quality assessment; (v) data extraction and (vi) synthesis.

2.1 Review Protocol

We developed a review protocol based on the guidelines and procedures as proposed in [28]. This protocol specifies the background for the review, research questions, search strategy, study selection criteria, data extraction and synthesis of the extracted data.

2.2 Inclusion and Exclusion Criteria

We only consider full papers in English from peer-reviewed journals, conferences and workshops. The review includes research studies that were published up to 2009. We exclude studies that do not relate to software engineering/development, software architecture and software quality analysis. We also exclude prefaces, articles in the controversial corner of journals, editorials, summaries of tutorials, panels and poster sessions. Furthermore, when several duplicated articles of a study exist in different versions that appear as journal papers, conference and workshop papers, we include only the most complete version of the study and exclude the others.

2.3 Search Process

We concentrate on searching in scientific databases rather than in specific books or technical reports, as we assume that the major research results in books and reports are also usually described or referenced in scientific papers. However, this does not prevent us from including a book as identified study when the book gives comprehensive descriptions of a certain relevant topic. 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, Wiley InterScience and ISI Web of Science. The searched results were also checked against a core set of studies within software architecture evolution and software quality analysis to ensure confidence in the comprehensiveness of search results.

The notion of evolvability is used in many different ways in the context of software engineering with many other closely-related quality attributes and synonyms such as flexibility, maintainability, adaptability and modifiability [13]. Therefore, the following search terms are used to find relevant studies:

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. The study selection process was performed through several steps: (i) Search in databases and conference proceedings to identify relevant studies; (ii) Exclude studies based on the basic exclusion criteria; (iii) Exclude studies based on titles and abstracts; and (iv) 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. After further filtering by reading titles and abstracts, 306 articles were left for full text screening. In the end, 54 articles were identified as primary studies. This search process was conducted in April 2009. An additional 4 papers were added after we had 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. This resulted in a total of 58 papers in the final list based on the inclusion and exclusion criteria.

2.4 Quality Assessment

To guide the interpretation of findings and determine the strength of inferences, we have, based on the quality assessment form [47], identified the following quality criteria for appraising the selected studies as these criteria indicate credibility of individual studies when synthesizing results. Among these quality criteria, the first two were used as the basis for including or excluding a study.

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 a description of the context in which the research was carried out;
4) The research designs address the aims of the study;
5) The study has a description of the data collection methods;
6) The data analysis is rigorous.

2.5 Data Extraction and Synthesis

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, research method descriptions, data collection and analysis, findings and conclusions. The data synthesis process includes identifying the main concepts from each study and analyzing how they relate to our review objectives.

3. RESULTS

Many studies promote and support software architecture evolvability through focusing on a particular technique or practice. Table 1 gives an overview of the studies according to publication channels, including the number of studies from each source along with the percentage of the total amount.

Table 1. Study Distribution per Publication Sources

Source	**Count**	**%**
Journal of Systems and Software	9	15.6
Books	6	10.4
Working IEEE/IFIP Conference on Software Architecture (WICSA)	5	8.6
Journal of Systems Engineering	4	6.9
International Conference on Software Engineering (ICSE)	4	6.9
IEEE International Conference on Software Maintenance (ICSM)	4	6.9
Journal of Information and Software Technology	2	3.5
IEEE International Computer Software and Applications Conference	1	1.7
ICSE Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK)	2	3.5
International Workshop on Principles of Software Evolution	2	3.5
International Conference on Quality Software (QSIC)	2	3.5
European Conference on Software Maintenance and Reengineering	2	3.5
Journal of Software Maintenance and Evolution	1	1.7
Journal of Systems Architecture	1	1.7
Journal of Computer Standards & Interfaces	1	1.7
Journal of Advanced Engineering Informatics	1	1.7
IEEE/ACM International Conference on Automated Software Engineering (ASE)	1	1.7
IEEE International Conference on Engineering of Complex Computer Systems	1	1.7
IEEE International Symposium on Requirements Engineering	1	1.7
International Conference on Software Reuse	1	1.7
International Software Metrics Symposium	1	1.7
ACM SIGSOFT software engineering notes	1	1.7
Conference of the Centre for Advanced Studies on Collaborative research	1	1.7
International Computer Software and Applications Conference	1	1.7
International Workshop on Economic-Driven Software Engineering Research	1	1.7
International Workshop on the Economics of Software and Computation	1	1.7
European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering	1	1.7
Total	**58**	**100**

The list of all the included studies is provided in the appendix of [12]. Of these 58 studies, 48 were published in leading journals, conferences or seminal books that are mostly cited in software engineering community. The representation of high quality and relevance of these studies and publication sources provides confidence in the overall quality assessment of the systematic review. After examining the research topics, data analysis and findings addressed in each study, we classify the identified primary studies into five main categories of themes, i.e. (i) techniques that support quality considerations during software architecture design, (ii) architectural quality evaluation, (iii) economic valuation, (iv) architectural knowledge management and (v) modeling techniques.

3.1 Quality Considerations Support during Software Architecture Design

Several studies focus on how software quality can be introduced and explicitly considered during software architecture design phase. They are classified into three groups based on their focused driving forces: quality attribute requirement-focused, quality attribute scenario-focused and influencing factor-focused.

3.1.1 Quality Attribute Requirement-Focused

Adaptability Evaluation Method (AEM) [S56][1] is an integral part of the Quality-driven Architecture Design and quality Analysis (QADA) methodology [36] specializing in the adaptability aspect. AEM captures the adaptability requirements that will be subsequently considered in the architecture design, provides guidelines on how to model adaptability in the architectural models, and analyzes the candidate architectures to ensure that adaptability requirements are met before system implementation.

Non-Functional Requirement (NFR) framework [S20] considers adaptability as a key quality attribute for evolving systems 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. Another concrete application example of using NFR framework describes an NFR approach in developing software system through using design patterns as potential adaptability enhancers [S19].

Bosch [S11] explicitly considers quality attributes during the design process and proposed a design method which examines three key phases, i.e. functionality-based architecture design, architecture assessment and architecture transformation.

Attribute-Driven Design (ADD) [S6] 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.

3.1.2 Quality Attribute Scenario-Focused

A systemized method for software architecture analysis throughout the processes of software design and development is described in [S18]. Architectural quality goals are expressed through scenarios that measure the goals, mechanisms that realize the scenarios, and analytic models that measure the results. 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.

Quality Attribute Workshop (QAW) [3] is a method that engages system stakeholders before the creation of the software architecture 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. 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. 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.

Active Reviews for Intermediate Designs (ARID) [S22] is a scenario-based assessment method for evaluating intermediate design or parts of an architecture. It is used to judge if the design of a partial architecture is appropriate for its intended purpose before the development of the complete architecture.

3.1.3 Influencing Factor-Focused

ArchDesigner [S55] is a quality-driven design approach for architectural design process. It attempts to best satisfy conflicting stakeholders' quality goals, architectural concerns and project constraints through using optimization techniques. This approach considers software architecture design problem as a global optimization problem because of the dependencies among different design decisions that need to be maintained, and the global constraints, e.g. stated project constraints that the selection of any design alternative must take into account.

Global analysis [S30] 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.

[S28] focuses on changeability and defines four aspects that have influence changeability, i.e. flexibility, agility, robustness and adaptability.

3.2 Quality Evaluation at the Software Architecture Level

Various business goals trigger architecture assessment activities [35], e.g. to evaluate and improve the architecture and its qualitative attributes, to identify any architectural drift and erosion, and to identify the risks related to a particular architecture. From evolution perspective, architecture evaluation is a preventive activity to delay the architectural decay and to limit the effect of software aging [46]. Several studies focus on software architecture evolvability evaluation and they are classified 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

[1] The references starting with S are the studies that were identified in the systematic review. A complete list of these included studies can be found in the appendix of a technical report [12].

knowledge of developers or consultants [1]. Attribute-Based Architectural Style (ABAS) [29] builds on architectural styles by explicitly associating with reasoning frameworks 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.

Empirically-Based Architecture Evaluation (EBAE) [34] 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 Architecture Level Modifiability Analysis [6] relates to software architecture comparison for optimal candidate architecture, and focuses on quantitatively measuring the stakeholders' views of the benefits and liabilities of software architecture candidates [S50]. The data collection is based on the knowledge, experiences and opinions of stakeholders. Any disagreements between the participating stakeholders are highlighted for further investigation.

A knowledge-based approach for assessing evolvability based on interviews with selected stakeholders, evaluates the evolutionary path of the software architecture during its lifecycle [S24]. The outcomes of the assessment include the current architecture overview, the main issues found and optionally recommendations for their resolutions. [S27] describes causes for changes during the lifecycle of a system as well as strategies to cope with changes. To understand the changes in the entire product development process, the authors suggest analyzing already finished projects to extract experiences on the most frequent changes in terms of sources of stimuli and cost of each change.

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 [37]. This is to avoid terminological ambiguities and conflicting interpretation of quality attributes. Software Architecture Analysis Method (SAAM) [S34, S35] 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 Tradeoff Analysis Method (ATAM) [S22, S36] evaluates software architectures in terms of quality attribute requirements. It is used to expose the risks, non-risks, sensitivity points and tradeoff points in the software architecture. An extension to ATAM is Holistic Product Line Architecture Assessment (HoPLAA) method [S43] for assessing product line architectures. It identifies risks at the core architecture level as well as the individual product architecture level. The notion of evolvability points is used to designate a sensitivity point or a tradeoff point that contains at least one variation point. Evolvability points ensure that quality attributes at individual product architecture level do not preclude core architecture quality attributes.

Architecture Level Modifiability Analysis (ALMA) [S9, S38, and S39] analyzes modifiability based on change scenarios that are used to capture future events the system needs to adapt to in its lifecycle. It can also assess risk and expose the boundaries of software architecture with respect to flexibility using complex scenarios [32].

Scenario-Based Architecture Reengineering (SBAR) [S8] considers multiple quality attributes, including development-oriented and operational-related ones. Another assessment method based on SAAM, ATAM and SBAR, evaluates evolvability of software product family architecture towards forthcoming requirements [S23]. The output includes the potential flaws and evolutionary path of the software.

3.2.3 *Metric-Based*

Several metrics have been proposed for evaluating evolvability. Ramil and Lehman proposed metrics based on implementation change logs [S45] and computation of metrics using the number of modules in a software system [S40]. Another set of metrics is based on software life span and software size [S52]. In [S48], 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 [16].

A process-oriented metric for software architecture adaptability is described in [S21], which analyzed the degree of adaptability through intuitive decomposition of goals and intuitive scoring for the goal-satisfying level of software architecture. As the method depends much on intuition and expert expertise, a quantitative metric-based approach that evaluates software architecture adaptability is proposed in [S41]. This approach supports decision-making in 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). Another approach to quantitatively analyze software evolution is using evolution ratio which is the amount of evolution in terms of software size, and evolution speed which is an indicator for the organization's capability for software system's evolution [S1].

A quality model provides a framework for quality assessment and aims to describe complex quality criteria through breaking them down into concrete subcharacteristics that are measured through metrics. Some well-known quality models are McCall's quality model [38], Dromey's quality model [21], Boehm's quality model [10], ISO 9126 [27] and FURPS quality model [25]. [S12] outlines a software evolvability model, in which subcharacteristics of software evolvability and corresponding measuring attributes are identified.

3.3 Economic Valuation in Determining the Level of Uncertainty

The uncertainties in software architecture evolution arise from, to certain extent, understanding how architectural decisions map onto quality attribute responses in terms of costs and benefits. Several approaches have been proposed to cope with uncertainty and mitigate risks in the investment. Cost Benefit Analysis Method (CBAM) [S46] 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. This method quantifies design decisions in terms of cost and benefits analysis to determine the level of uncertainty and decides how to prioritize changes to architecture, based on perceived difficulty and utility. A related economics-driven method is Architecture Improvement Workshop (AIW) (http://www.sei.cmu.edu/architecture/products_services/aiw.html) which values architectural decisions in relation to quality attributes.

Software architecture decisions carry economic value in form of real options [2, 45]. Options offer flexibility and consider architectural evolution over time [S4, S25]. An approach that considers cost, value and alignment with business goals to support architectural evolution is described in [S31]. This approach guides the selection of design patterns, the elicitation of architecturally significant requirements, and the valuation of architecture in terms of design decisions with multiple quality-attribute viewpoints. Another application of real options theory is described in [S5], which provides insights into architectural flexibility and investment decisions related to the evolution of software systems. This approach examines a set of probable changes as well as their added value, e.g. accumulated savings through enduring the change without violating architectural integrity; supporting future growth; and capability of responding to competitive forces and changing market conditions.

Another way to address economic valuation is through estimating the required effort for system modification to accommodate future changes, e.g. maintenance cost prediction [S7] calculates 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.

Given particular schedule constraints, a model-based approach [S44] strategically determines an appropriate degree of architectural flexibility through four strategic elements, i.e. feature prioritization, schedule range estimation, core capability determination and architecture flexibility determination, thus to mitigate the risk of violating schedule, cost and quality constraints. [S49] is another example that treats evolvability of software design using the value of strategic flexibility.

A conceptual approach to quantifying a system's life cycle value is proposed in [S14] to facilitate adaptability to changing circumstances and stakeholder preferences. This approach is based on several key parameters for quantifying the perceived value to a system's stakeholders.

3.4 Architectural Knowledge Management

Architectural knowledge comprises architecture design, design decisions, assumptions, context, and other factors that together shape software architecture. An explicit representation of architectural knowledge is helpful for evolving quality systems and assessing future evolutionary capabilities of a system [30]. Apart from using change scenarios and change cases to explicitly model variability and describe the future evolutionary capabilities, it is also useful to explicitly model invariability assumptions, i.e. things that are assumed will not change [S37]. Assumptions are design decisions and rationale that are made out of personal experience and background, domain knowledge, budget constraints and available expertise. This information can be used to provide additional what-if scenarios for software architecture assessment, i.e. what if a certain assumption proves to be invalid. In addition, explicit representation of the traceability between the software architecture evolution and the early-made assumptions augments design decisions in the face of uncertainties when predicting the future user requirement changes. A relevant method is Recovering Architectural Assumptions Method (RAAM) [S47] that makes the assumptions explicit through recapitulating historical information of software system evolution.

To assess architectural design erosion [49], an architecture assessment model objectively measures the extent of deviation in terms of functional and structural divergence [S10]. To track software evolution, the loss of system functionality and architectural structure as a software system evolves is represented through functional and structural erosion indicators. Documenting design rationale is another approach that is used to maintain and evolve architectural artifacts [S54] in order to allow unanticipated changes in the software without compromising software integrity and to evolve in a controlled way [8]. The concept of architectural constraints is introduced in [S29] to generalize architectural styles, patterns and similar concepts, based on the conjecture that architectural constraints influence the quality of architectural design process and the improvement of software quality. In order to provide support for capturing quality attribute knowledge, design decisions for quality attributes and their rationale, several tools have been developed [S2, S3, S15, S16, S17, S26, S32, and S33]. A comparative study of these architecture knowledge management tools is detailed in [S53].

An initial work on improving architecture evaluation activities for pattern oriented systems is to improve software architecture evaluation through mining patterns [S58] to systematically extract and document architecturally significant information.

3.5 Modeling Techniques

Several modeling techniques enable software architecture evolvability. One of them is Business Rule Model [S57] that captures and specifies business rules, and relates them to the metamodel level of software design elements through a Link Model. As business rules have impact on software and business process, explicit consideration and modeling of business rules facilitate the improvement of software evolvability. Another way to enable evolvability is a model-based approach for modeling the traceability links through considering the relations between requirements, architectural elements and implementation [S13]. A quality-driven software reengineering model [S51] addresses the evolution of system requirements and software architecture, by adopting NFR Framework [16] and the concept of soft goals to support the systematic modeling of the design rationale through a soft-goal interdependency graph.

To capture and assess software architectures for evolution and reuse, a framework for modeling relevant information and architectural views for reengineering, analyzing, and comparing software architectures is proposed [S42]. 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.

4. DISCUSSIONS

This section discusses implications for research and practice, summarizes the principle findings of the systematic review, and discusses the validity threats of this review.

4.1 Implications for Research

1. Potential to further improve the value and applicability of research ideas The systematic review provides us a perspective of where the field of software evolution and software architecture evolvability stands today. One indicator of technology maturity is Redwine-Riddle model [42], which identifies six typical phases for technology maturation, i.e. basic research, concept formulation, development and extension, internal enhancement and exploration, external enhancement and exploration, and popularization. We examine the maturity phase of ideas and concepts, as well as the status of their applications by looking into the research methods used in each study's validation. A distribution of the studies per research method is shown in Table 2. Only one-fourth of the studies have extended their approaches to other domains and use the approach for industrial problems, indicating accomplishment of the internal enhancement and exploration phase. Two surveys were conducted on practitioners in companies. Most of the case studies are single-case, with 20 studies done in projects in industry and 13 studies in academic settings. This implies a potential for future research to further improve the value and applicability of these research ideas.

Table 2. Study Distribution per Research Method

Research Method	Number	%
Single-case in Industry	20	34.5
Single-case in Academia	13	22.4
Multiple-case	16	27.6
Theoretical Reasoning	7	12.1
Survey	2	3.4
Total	**58**	**100**

2. Potential areas for further research With respect to the research topics of the studies, we illustrate the distribution of the studies per architecture-centric activity [4] in Figure 1. The lifecycle activities of architecture-centric development include:

A. Creating the business case for the system;

B. Understanding the requirements;

C. Creating or selecting the software architecture;

D. Documenting and communicating the software architecture;

E. Analyzing or evaluating the software architecture;

F. Implementing the system based on the software architecture;

G. Ensuring that implementation conforms to the software architecture.

Several studies address multiple architecture activities [S21, S51, S58]. Many studies address architecture analysis and evaluation. The recognition of a strong connection between architecture and business value as well as the emergence of tool support for architecture documentation signals the progress in the field of software architecture evolution. However, it is interesting to note that two architecture activities, i.e. implementation based on the architecture and conformance checking, are addressed by comparatively fewer studies. This seems to be in accordance with the potential areas for improvement that were identified in a recent IEEE Software article [19], i.e. object-oriented programming vs. architecture, conformance checking.

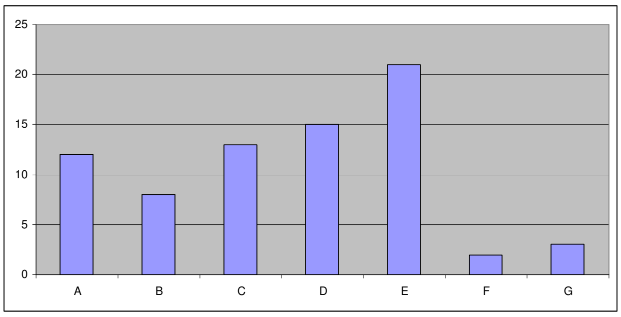

Figure 1. Study Distribution per Architecture-Centric Activity

3. Combination of appropriate techniques with a lifecycle view Each of the techniques and approaches identified in the review 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 [33]. 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. Economic valuation methods complement with details on architectural decisions' business consequences. We see the initiative in research community to combine appropriate techniques for software architecture evolution [23, 40]. As evolvability needs to be addressed and maintained throughout the complete software lifecycle, combination of appropriate techniques with a lifecycle view poses a future research theme that is necessary for software systems to cope with diverse types of influencing factors.

4. Software process model facilitating software architecture evolution Software engineering is a field with several disciplines relevant to software architecture evolution, e.g. the exploration and introduction of software development paradigms, such as agile software development [20]. None of the identified studies address this aspect. This might be due to the assumption of the lack of focus on architecture in agile development methods [22]. However, Rajlich [41] describes that 'the new Agile paradigm brings a host of new topics into the forefront of software engineering research'. Researchers have started to explore the interplay between software process improvement and software architecture [17, 39]. Continuing to investigate practices and techniques in software process models that facilitate software architecture evolvability remains an interesting research theme.

4.2 Implications for Practice

As indicated in implications for research, each technique and approaches identified in the review has its specific context that it is appropriate for. The main consideration for practitioners is thus to use this review as a source in searching for relevant studies, compare the settings to their own context, adopt and tailor appropriate methods in their software development lifecycle. In this process, practitioner can find multiple dimensions of factors that exert influence on software architecture evolvability, e.g. (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 [9], and business models evolve [48]. From technology perspective, many unknown, uncontrollable technological and environmental constraints outweigh design principles [26]. Assumptions shaping software architectures [31] 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 [5]. 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. We believe that understanding of these influencing factors and causes for changes will benefit practitioners in tailoring the approaches so that they can fit more easily into their own existing lifecycle models.

4.3 Principle Findings

The goal of this review was to identify a holistic view of the existing studies in analyzing and achieving software evolvability at architectural level. 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.

Besides, we have discussed the implications for research and practice.

4.4 Validity

One possible threat of the review is bias in the selection of publications and data extraction. This is addressed through specifying a research protocol that defines the research questions, inclusion and exclusion criteria, search strategy and strategy for data extraction. The research protocol and the identified publications have been reviewed by several researchers to minimize the risk of exclusion of relevant studies. Besides, additional reference checking of the identified studies was conducted to guarantee a representative set of studies for the review.

5. CONCLUSIONS

The systematic review of studies in promoting software evolvability at architectural level identified 58 primary studies. They 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. The main implications for research include further improvement of the value and applicability of research ideas, and integrate appropriate approaches with a lifecycle viewpoint. The main implications for practice is the need to understand the causes and factors that exert influence on software architecture evolvability, and the need to tailor appropriate methods to suit their application context.

6. REFERENCES

[1] Avritzer, A., and Weyuker, E.J.: 'Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews', Empirical Software Engineering, 1999, 4, (3), pp. 199-215

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

[3] 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.

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

[5] Bass, L., Nord, R., Wood, W., Zubrow, D., and Ozkaya, I.: 'Analysis of architecture evaluation data', Journal of Systems and Software, 2008, 81, (9), pp. 1443-1455

[6] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H.: 'Architecture-level modifiability analysis (ALMA)', Journal of Systems and Software, 2004, 69, (1-2), pp. 129-147

[7] Bennett, K.: 'Software evolution: past, present and future', Information and Software Technology, 1996, 38, (11), pp. 673-680

[8] Bennett, K.H., and Rajlich, V.T.: 'Software maintenance and evolution: a roadmap', (ACM NY, USA, 2000, edn.), pp. 73-87

[9] Bhattacharya, S., and Perry, D.E.: 'Architecture assessment model for system evolution', (Inst. of Elec. and Elec. Eng. Computer Society, 2007.

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

[11] Borne, I., Demeyer, S., and Galal, G.H.: 'Object-oriented architectural evolution', LECTURE NOTES IN COMPUTER SCIENCE, 1999, 1743, pp. 57-64

[12] Breivold, H.P., and Crnkovic, I.: 'A Survey of Software Evolvability', MRTC report ISSN 1404-3041 ISRN MDH-

MRTC-239/2009-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, September, 2009
[13] Breivold, H.P., Crnkovic, I., and Eriksson, P.J.: 'Analyzing Software Evolvability', compsac, 2008.
[14] Cai, Y., and Huynh, S.: 'An evolution model for software modularity assessment', ICSE Workshop on Software Quality 2007.
[15] Christian, D.R.: 'Continuous evolution through software architecture evaluation: a case study', J. Softw. Maint. Evol.: Res. Pract, 2006, 18, pp. 351-383
[16] Chung, L.: 'Non-Functional Requirements in Software Engineering' (Springer, 2000.)
[17] Clements, P., Ivers, J., Little, R., Nord, R., Stafford, J., and File, H.: 'Documenting Software Architectures in an Agile World', Technical Note CMU/SEI-2003-TN-023, 2003.
[18] Clements, P., Kazman, R., and Klein, M.: 'Evaluating Software Architectures: Methods and Case Studies' (Addison-Wesley, 2002.)
[19] Clements, P., and Shaw, M.: '"The Golden Age of Software Architecture" Revisited', IEEE Software 2009.
[20] Cockburn, A.: 'Agile software development' (Addison-Wesley Boston, MA, 2002.)
[21] Dromey, R.G.: 'Cornering the Chimera', IEEE Software, 1996, 13, (1), pp. 33-43
[22] Dybå, T., and Dingsøyr, T.: 'Empirical studies of agile software development: A systematic review', Information and Software Technology, 2008
[23] Falessi, D., Capilla, R., and Cantone, G.: 'A value-based approach for documenting design decisions rationale: a replicated experiment', ACM NY, USA, 2008, pp. 63-70
[24] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: 'Design patterns: elements of reusable object-oriented software' (1995.)
[25] Grady, R.B., and Caswell, D.L.: 'Software metrics: establishing a company-wide program' (Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1987.)
[26] Graham, T.C.N., Rick, K., and Chris, W.: 'Agility and Experimentation: Practical Techniques for Resolving Architectural Tradeoffs'. Proc. Proceedings of the 29th international conference on Software Engineering2007.
[27] ISO9126: 'ISO/IEC 9126-1, International Standard, Software Engineering. Product Quality – Part 1: Quality Model'
[28] Kitchenham, B.: 'Procedures for performing systematic reviews', Keele University TR/SE-0401/NICTA Technical Report 0400011T, 2004, 1
[29] Klein, M., Kazman, R., Bass, L., Carriere, J., Barbacci, M., and Lipson, H.: 'Attribute-Based Architecture Styles' (Kluwer, BV Deventer, The Netherlands. 1999)
[30] Kruchten, P., Lago, P., and van Vliet, H.: 'Building up and reasoning about architectural knowledge', LECTURE NOTES IN COMPUTER SCIENCE, 2006
[31] Lago, P., and van Vliet, H.: 'Explicit assumptions enrich architectural models', ICSE 2005.
[32] Lassing, N., Rijsenbrij, D., and van Vliet, H.: 'On software architecture analysis of flexibility, Complexity of changes: Size isn't everything', 2nd Nordic Software Architecturee Workshop 1999, pp. 1103-1581
[33] Lassing, N., Rijsenbrij, D., and van Vliet, H.: 'How well can we predict changes at architecture design time?' Journal of Systems and Software, 2003, 65, (2), pp. 141-153
[34] Lindvall, M., Tvedt, R.T., and Costa, P.: 'An Empirically-Based Process for Software Architecture Evaluation', Empirical Software Engineering, 2003, 8, (1), pp. 83-108
[35] Maccari, A.: 'Experiences in assessing product family software architecture for evolution', ACM NY, USA, 2002, pp. 585-592
[36] Matinlassi, M.: 'Quality-driven software architecture model transformation', WICSA 2005.
[37] Mattsson, M., Grahn, H., and Mårtensson, F.: 'Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability', QoSA 2006.
[38] 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.)
[39] Nord, R.L., and Tomayko, J.E.: 'Software architecture-centric methods and agile development', IEEE Software, 2006, pp. 47-53
[40] Nord, R.L., Wood, W.G., and Clements, P.C.: 'Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method', Technical Note CMU/SEI-2004-TN-017
[41] Rajlich, V.: 'Changing the paradigm of software engineering', Communications of the ACM 2006.
[42] Redwine Jr, S.T., and Riddle, W.E.: 'Software technology maturation', (IEEE Computer Society Press Los Alamitos, CA, USA, 1985, edn.), pp. 189-200
[43] Rowe, D., and Leaney, J.: 'Evaluating evolvability of computer based systems architectures-an ontological approach', ECBS 1997, pp. 24-28
[44] Rowe, D., Leaney, J., and Lowe, D.: 'Defining systems evolvability-a taxonomy of change', Change, 1994, pp. 541-545
[45] 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, 1999, pp. 215–262
[46] Tonu, S.A., Ashkan, A., and Tahvildari, L.: 'Evaluating architectural stability using a metric-based approach', CSMR 2006.
[47] Tore, D., Torgeir, D., and yr: 'Empirical studies of agile software development: A systematic review', Inf. Softw. Technol., 2008, 50, (9-10), pp. 833-859
[48] Wan-Kadir, W.M.N., and Loucopoulos, P.: 'Relating evolving business rules to software design', Journal of Systems Architecture, 2004, 50, (7), pp. 367-382
[49] van Gurp, J., and Bosch, J.: 'Design erosion: problems and causes', Journal of Systems and Software, 2002, 61, (2), pp. 105-119
[50] Weiderman, N.H., Bergey, J.K., Smith, D.B., and Tilley, S.R.: 'Approaches to Legacy System Evolution', CMU/SEI-97-TR-014, 1997.