

# Architecting Software for Evolvability: A Systematic Review

Hongyu Pei Breivold<sup>1</sup>, Ivica Crnkovic<sup>2</sup>, Magnus Larsson<sup>1</sup>

<sup>1</sup>*ABB Corporate Research, Industrial Software Systems, 721 78 Västerås, Sweden  
{hongyu.pei-breivold, magnus.larsson}@se.abb.com*

<sup>2</sup>*Mälardalen University, 721 23 Västerås, Sweden  
ivica.crnkovic@mdh.se*

## Abstract

**Context:** Software evolvability describes a software system's ability to easily accommodate future changes. It is a fundamental characteristic for making strategic decisions, and increasing economic value of software. For long-lived systems, there is a need to address evolvability explicitly during the entire software lifecycle in order to prolong the productive lifetime of software systems. For this reason, many research studies have been proposed in this area both by researchers and industry practitioners. These studies comprise a spectrum of particular techniques and practices, covering various activities in software lifecycle. However, no systematic review has been conducted previously to provide an extensive overview of software architecture evolvability research.

**Objective:** In this work, we present such a systematic review of architecting for software evolvability. The objective of this review is to obtain an overview of the existing approaches in analyzing and improving software evolvability at architectural level, and investigate impacts on research and practice.

**Method:** The identification of the primary studies in this review was based on a pre-defined search strategy and a multi-step selection process.

**Results:** Based on research topics in these studies, we have identified five main categories of themes: (i) techniques supporting quality consideration during software architecture design, (ii) architectural quality evaluation, (iii) economic valuation, (iv) architectural knowledge management, and (v) modeling techniques. A comprehensive overview of these categories and related studies is presented.

**Conclusion:** The findings of this review also reveal suggestions for further research and practice, such as (i) it is necessary to establish a theoretical foundation for software evolution research due to the fact that the expertise in this area is still built on the basis of case studies instead of generalized knowledge; (ii) it is necessary to combine appropriate techniques to address the multifaceted perspectives of software evolvability due to the fact that each technique has its specific focus and context that it is appropriate for in the entire software lifecycle.

*Keywords: Software evolvability, Systematic review, Software architecture*

# 1. Introduction

It has long been recognized that, for long-lived industrial software, the largest part of lifecycle costs is concerned with the evolution of software to meet changing requirements [3]. To keep up with new business opportunities, the need to change software on a constant basis with major enhancements within a short timescale puts critical demands on the software system's capability of rapid modification and enhancement. Lehman [18] describes two perspectives on software evolution: “*what and why*” versus “*how*”. The “*what and why*” perspective studies the nature of the software evolution phenomenon and investigates its driving factors and impacts. The “*how*” perspective studies the pragmatic aspects, i.e. technology, methods and tools that provide the means to control software evolution. In this research, we focus on the “*how*” perspective of software evolution.

The term evolution reflects “*a process of progressive change in the attributes of the evolving entity or that of one or more of its constituent elements*” [20]. Specifically, software evolution relates to how software systems change over time [34]. One of the principle challenges in software evolution is therefore the ability to evolve software over time to meet the changing requirements of its stakeholders [25], and to achieve cost-effective evolution. In this context, software evolvability has emerged 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*” [29].

The ever-changing world makes evolvability a strong quality requirement for the majority of software architectures [5, 28]. The inability to effectively and reliably evolve software systems means loss of business opportunities [4]. Based on our experiences and observations from various cases in industrial contexts (for example [S15]<sup>1</sup>, [S34], [16]), we have noticed that industry starts to have serious considerations related to evolvability beyond maintainability. From these studies, we also witness 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. Software systems must often reflect these changes to adequately fulfill their roles and remain relevant to stakeholders. Evolvability was therefore identified in these cases as a very important quality attribute that must be continuously maintained during their lifecycle. As software evolvability is a fundamental element for an efficient implementation of strategic decisions, and increasing economic value of software [7, 33], for such long-lived systems, there is a need to address evolvability explicitly during the entire lifecycle and thus prolong the productive lifetime of software systems.

Analyzing and improving software evolution can be done through various ways; e.g. analyzing release histories, source code, and software architecture level analysis. Our research focuses on the *architectural level* analysis for two reasons. Firstly, the foundation for any software system is its

---

<sup>1</sup> The references starting with S are the studies that were identified in the systematic review. A complete list of these studies can be found in the appendix.

architecture, which allows or precludes nearly all of the quality attributes of the system [S30]. For instance, a system without an adaptable architecture will degenerate sooner than a system based on an architecture that takes changes into account [10]. Secondly, the architecture of a software system describes its high level structure and behavior, thus, software architecture exposes the dimensions along which a system is expected to evolve [11] and provides basis for software evolution [23]. Therefore, architecture evolution permits planning and system restructuring at a high level of abstraction where quality and business tradeoffs can be analyzed [S39].

Many research studies focus on how to analyze and improve software evolvability, using a particular technique or practice. However, no systematic review of software architecture evolvability research has been published previously to describe the wide spectrum of results in these studies. The main objective of our research is therefore to systematically select and review published literature, and present a holistic overview of the existing studies in analyzing and achieving software evolvability at architectural level. Secondary objectives are (i) *to bring practitioners up to date to the state of research themes* that have been actively pursued by the research community in software evolvability architecting, and quickly identifying relevant studies that suit their own needs; (ii) *to help the research community to identify challenges and research gaps* that require further exploration. Concretely, we stated the following research questions:

- 1) What approaches have been reported regarding the analysis and achievement of software evolvability at the architectural level?
- 2) What are the main research themes covered in the scientific literature regarding analysis and achievement of evolvability-related quality attributes?
- 3) What are the main focus and application contexts of these approaches, along with their relevance to software evolvability?
- 4) What is the impact of the studies to research community and practice?

The remainder of the article is structured as follows. Section 2 describes the research method used in this review. Section 3 presents overview information of the studies included in our systematic literature review (SLR). Section 4 presents the results of the review in five main categories of themes with detailed description of relevant studies and analysis of their relevance to evolvability. Section 5 discusses principle findings of the review along with its impact on research and practice, as well as validity threats of the review. Section 6 concludes the paper.

## **2. Research Method**

This research was undertaken as a systematic review [14] which is a formalized and repeatable process to document relevant knowledge on a specific subject area for assessing and interpreting all available research related to a research question. The research 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 synthesis. These stages are detailed in the following subsections.

## 2.1 Review Protocol

We formulated a review protocol based on the systematic literature review guidelines and procedures [14]. This protocol specifies the background for the review, research questions, search strategy, study selection criteria, data extraction and synthesis of the extracted data. The protocol was mainly developed by one author and reviewed by the other authors to reduce bias. The background and the research questions are described in the Introduction section, while other elements are described below.

## 2.2 Inclusion and Exclusion Criteria

The goal of setting up criteria is to find all relevant studies in our research. We consider full papers in English from peer-reviewed journals, conferences and workshops published up to and including the first two quarters of 2010. We did not set a lower boundary on the year of publication because we intended to include all relevant studies that are stored in databases over the years. We exclude studies that do not explicitly relate to software evolution, analysis of software architecture, and software quality related to software evolution. We also exclude prefaces, editorials, and summaries of tutorials, panels and poster sessions. Furthermore, when several duplicated articles of a study exist in different versions that appear as books, journal papers, conference and workshop papers, we include only the most complete version of the study and exclude the others. A summary of the inclusion and exclusion criteria for this review is presented in Table 1. Note that a study must satisfy all inclusion criteria, and not satisfy any of the exclusion criteria.

Table 1. Inclusion and exclusion criteria

<b>Inclusion Criteria</b>
English peer-reviewed studies that provide answers to the research questions.
Studies that focus on software evolution.
Studies that focus on software architecture analysis and/or software quality analysis related to software evolvability.
Studies are published up to and including the first two quarters of 2010.
<b>Exclusion Criteria</b>
Studies are not in English.
Studies that are not related to the research questions.
Studies in which claims are non-justified or ad-hoc statements instead of based on evidence.
Duplicated studies.

## 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 an identified study if the book gives comprehensive descriptions of a certain relevant topic. For instance, the Architecture Tradeoff Analysis Method (ATAM) was described in a conference paper [12] and it was also thoroughly explained in a book [S30]. We have therefore included the book as a selected study.

The searched electronic databases include:

- ACM Digital Library (<http://portal.acm.org>)
- Compendex (<http://www.engineeringvillage.com>)
- IEEE Xplore (<http://www.ieee.org/web/publications/xplore/>)
- ScienceDirect – Elsevier (<http://www.elsevier.com>)
- SpringerLink (<http://www.springerlink.com>)
- Wiley InterScience (<http://www3.interscience.wiley.com>)
- ISI Web of Science (<http://www.isiknowledge.com>).

These databases were chosen as they provide the most important and with highest impact full-text journals and conference proceedings, covering the fields of software quality, software architecture and software engineering in general. After an initial search of these databases, we did an additional reference scanning and analysis in order to find out whether we have missed anything, thus to guarantee that we have selected a representative set of studies. 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 alternative words such as *flexibility*, *maintainability*, *adaptability* and *modifiability*. Therefore, we consider these words in the list of search terms. In addition, we have, in our earlier work [S15], outlined a software evolvability model and identified subcharacteristics that are of primary importance for a software system to be evolvable. The identified subcharacteristics are a union of quality characteristics that are relevant for characterization of evolution of long-lived software-intensive systems during their life cycle, comprising *analyzability*, *architectural integrity*, *changeability*, *extensibility*, *portability*, *testability* and *domain-specific attributes*. These evolvability subcharacteristics thus, also provide input and motivate the search terms that we use in the research when searching for relevant studies.

Among evolvability subcharacteristics, portability and testability are not explicitly considered as search terms for the review, as we have in the preliminary search found that they are quite often pertained to maintainability, adaptability and flexibility. Domain-specific attribute comprises quality characteristics that are specific for a particular domain, and is considered too general to be used as a search term. The remaining subcharacteristics, *analyzability*, *changeability* and *extensibility* are included as search terms. Therefore, the following search terms are used to find relevant studies, and all these search terms were combined by using the Boolean OR operator:

- 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
- S8: software architecture AND analyzability

The selection of studies was performed through a multi-step process:

- (i) Search in databases to identify relevant studies by using the search terms;
- (ii) Exclude studies based on the exclusion criteria;
- (iii) Exclude irrelevant studies based on analysis of their titles and abstracts;
- (iv) Obtain primary studies based on full text read.

Figure 1 shows the search process and the number of publications identified at each stage. Duplicate publications were removed. We performed the search process at two points in time, i.e. one in August 2009, and the other one in the end of August 2010, with the intention to cover the latest results of publications in 2009 and 2010. In the first search process, the search strategy identified a total of 3036 publications that we entered into the tool EndNote<sup>2</sup>, which was also used in the subsequent steps for reference storage and sorting. These publications were checked against the inclusion and exclusion criteria. Irrelevant publications were removed and this resulted in 731 remaining publications. After further filtering by reading titles and abstracts, 306 publications were left for full text screening to ensure that the contents indeed relate to the topic of software architecture evolution. In the end, 58 studies were identified as primary studies after the first search process. After we had performed a complementary search in the end of August, 2010, 24 new papers were added. This resulted in a total of 82 studies in the final list, covering the publications up to and including the first two quarters of 2010. We explain the relative high increase of the studies as: 1) inclusion of studies from 2009 and 2010 (since several studies from 2009 were not available in the database in the first search), and 2) the increased interest in the topic.

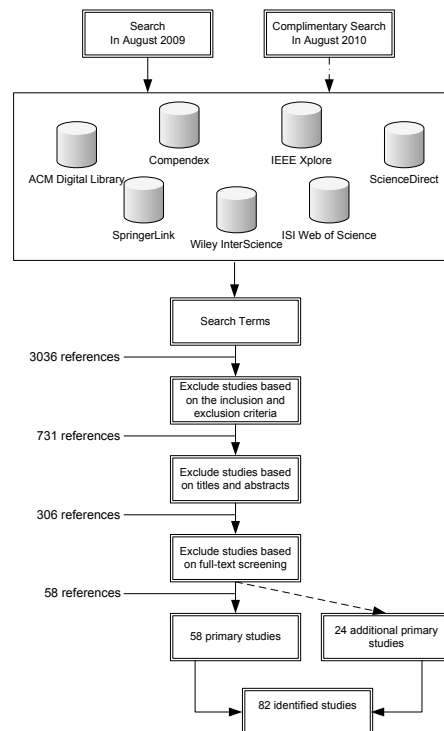


Figure 1. Stages of the Search Process

<sup>2</sup> [www.endnote.com](http://www.endnote.com)

## 2.4 Quality Assessment

To guide the interpretation of findings in the included studies, and determine the strength of inferences, we used the following quality criteria for appraising the selected studies. These criteria indicate the credibility of an individual study when synthesizing results:

- 1) The data analysis of the study is rigorous and based on evidence or theoretical reasoning instead of non-justified or ad-hoc statements;
- 2) The study has a description of the context in which the research was carried out;
- 3) The aims of the study are supported by the design and execution of research;
- 4) The study has a description of the research method used for data collection;

These criteria ascertained our confidence in the credibility of a particular identified study and its relevance for data synthesis in the review.

## 2.5 Data Extraction and Synthesis

The data extraction and synthesis process was carried out by reading each of the 82 papers thoroughly and extracting relevant data, which were managed through bibliographical management tool EndNote and Excel spreadsheets. In order to keep information consistent the data extraction for the 82 studies was driven by a form shown in Table 2. For the data synthesis, we inspected the extracted data for similarities in order to define how results could be encapsulated. The results of the synthesis will be described in the subsequent sections.

Table 2. Data Extraction for Each Study

Extracted Data	Description
Identity of study	Unique identity for the study
Bibliographic references	Author, year of publication, title and source of publication
Type of study	Book, journal paper, conference paper, workshop paper
Focus of the study	Main topic area, concepts, motivation and objective of the study
Research method used for data collection	Included technique for the design of the study, e.g. case study, survey, experiment, interview to obtain data, observation
Data analysis	Qualitative or quantitative analysis of data
Application context	Description of the context and application settings of the study, e.g. domain, academic or industrial settings
Constraints and limitations	Identified constraints and limitations in the application of an approach as well as the identified areas for future research
Architecture-centric activity	Indicating the architecture-centric activity on which the study is focused, e.g. business case, creating architecture, documenting architecture, analyzing architecture, etc.
Software lifecycle	The phase of software lifecycle covered in the study

## 3. Overview of the Included Studies

A list of all the selected studies is provided in the appendix. The section describes these studies with respect to their sources of publication and citation status which are also indicators on the quality and their impact. A temporal view and research communities that are active in the field of software architecture evolution are presented as well.

### 3.1 Data Sources

Most of these 82 studies were published in leading journals, conferences or seminal books that belong to the most cited publication sources in software engineering community. Table 3 gives an overview of the distribution of the studies based on their publication channels, along with the number of studies from each source. All the studies fulfill the criteria for quality assessment as described in the previous section. In addition, the impact factor<sup>3</sup> of the publication sources represents also the degree of high quality and potential impact of these studies, and provides confidence in the overall quality assessment of the systematic review. This is also indicated in the citation status described in the next subsection.

Table 3. Study Distribution per Publication Sources

Source	Count
Journal of Systems and Software (JSS)	14
Working IEEE/IFIP Conference on Software Architecture (WICSA)	8
Books	5
International Conference on Software Engineering (ICSE)	5
Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK)	5
IEEE International Conference on Software Maintenance (ICSM)	4
Journal of Information and Software Technology (IST)	4
Journal of Systems Engineering	4
International Conference on Quality Software (QSIC)	3
International Workshop on Principles of Software Evolution (IWPSE)	2
IEEE/ACM International Conference on Automated Software Engineering (ASE)	2
European Conference on Software Maintenance and Reengineering	2
IEEE International Conference on Engineering of Complex Computer Systems	2
Journal of Software Maintenance and Evolution	1
Journal of Systems Architecture	1
Journal of Computer Standards & Interfaces	1
Journal of Advanced Engineering Informatics	1
Journal of Software: Practice and Experience	1
IEEE International Computer Software and Applications Conference	1
IEEE International Symposium on Requirements Engineering	1
IEEE Software	1
International Conference on Software Engineering Advances	1
International Conference on Information Science and Applications (ICISA)	1
International Conference on Research Challenges in Information Science	1
International Conference on Software Reuse	1
International Software Metrics Symposium	1
ACM SIGSOFT software engineering notes	1
Conference of the Centre for Advanced Studies on Collaborative research	1
International Conference and Workshops on Engineering of Computer-Based Systems (ECBS)	1
ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing	1

<sup>3</sup> For instance, based on the search results (performed on 22<sup>nd</sup> of September, 2010) in respective journal web sites, JSS has impact factor of value 1.34, JST with value of 1.82, Journal of Advanced Engineering Informatics of value 1.73.



International Computer Software and Applications Conference	1
International Workshop on Economic-Driven Software Engineering Research	1
International Workshop on the Economics of Software and Computation	1
European software engineering conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering	1
World Congress on Computer Science and Information Engineering (CSIE)	1
<b>Total</b>	<b>82</b>

### 3.2 Citation Status

Table 4 provides an overview of the citation rates of the included studies. These numbers are obtained from Google Scholar<sup>4</sup>. The data presented here only gives a rough indication of citation rates, and are not meant for comparison among studies. 35 studies have been cited by less than 10 other sources. Among these 35 studies, 22 are published in 2009 and 2010, so it is not expected that they can reach a higher citation number in such a short period. Almost half of the studies (38 studies) have been cited by more than 20 other sources. Thirteen studies have very high citation rates with more than 80 other sources.

Table 4. Status of Citation Rate in Detail

Cited by	< 10	10 - 20	20 - 30	30 - 40	40 - 50	50 - 60	60 - 70	70 - 80	> 80
<b>No. of Studies (Total 82)</b>	35	9	10	1	6	4	2	2	13

We can see that, in general; the citation rates of the studies are quite high, which is also an indicator on the high quality and impact of the studies. We expect that the number of citations will grow since most of the papers have been published in the last six years (see section 3.3). The most cited studies (cited by more than 60 other sources) are summarized in Table 5. The first five studies are books, and the rest are papers in journals and conferences.

Table 5. Most Cited Studies

Ranking	Studies	Titles
1	S8	L. Bass, P. Clements, R. Kazman, Software architecture in practice, Addison-Wesley Professional, 2003.
2	S27	L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, Non-functional requirements in software engineering: Springer, 2000.
3	S13	J. Bosch, Design and use of software architectures: adopting and evolving a product-line approach, Addison-Wesley Professional, 2000.
4	S30	P. Clements, R. Kazman, M. Klein, Evaluating software architectures: methods and case studies, Addison-Wesley, 2006.
5	S42	C. Hofmeister, R. Nord, D. Soni, Applied Software Architecture: A Practical Guide for Software Designers, Addison-Wesley Professional, 2000.
6	S47	R. Kazman, L. Bass, G. Abowd, M. Webb, SAAM: a method for analyzing the properties of software architectures, International Conference on Software Engineering, pp. 81-90, 1994.
7	S48	R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 68-78, 1998.
8	S56	M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, W. M. Turcki, Metrics and laws of software evolution-the nineties view, 4th International Software Metrics Symposium 1997.
9	S50	M. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, H. Lipson, Attribute-based architecture styles, Working IEEE/IFIP Conference on Software Architecture (WICSA) 1999.

<sup>4</sup> <http://scholar.google.se/> accessed on 4<sup>th</sup> of September, 2010

10	S72	K. J. Sullivan, W. G. Griswold, Y. Cai, B. Hallen, 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.
11	S11	P. Bengtsson, N. Lassing, J. Bosch, H. van Vliet, Architecture-level modifiability analysis (ALMA), Journal of Systems and Software, vol. 69, pp. 129-147, 2004.
12	S46	R. Kazman., J. Asundi, M. Klein, Quantifying the costs and benefits of architectural decisions, 23rd International Conference on Software Engineering, 2001.
13	S9	P. Bengtsson, J. Bosch, Architecture level prediction of software maintenance, 3rd European Conference on Software Maintenance and Reengineering (CSMR), pp. 139-147, 1999.
14	S10	P. Bengtsson, J. Bosch, Scenario-based software architecture reengineering, International Conference on Software Reuse, pp. 308-317, 1998.
15	S81	W. M. N. Wan-Kadir, P. Loucopoulos, Relating evolving business rules to software design, Journal of Systems Architecture, vol. 50, pp. 367-382, 2004.
16	S53	N. Lassing, P. Bengtsson, H. van Vliet, J. Bosch, Experiences with ALMA: Architecture-Level Modifiability Analysis, Journal of Systems and Software, vol. 61, pp. 47-57, 2002.
17	S45	A. Jansen, J. Van der Ven, P. Avgeriou, D. K. Hammer, Tool support for architectural decisions, Working IEEE/IFIP Conference on Software Architecture (WICSA) 2007.

### 3.3 Temporal View

Looking at the studies by year of publication as shown in Figure 3, we notice in the trend curve an increasing number of publications in the area of software architecture evolution since 1999. (Note that for year 2010, the review only covers the registered publication in the databases until the first two quarters.) We also notice that all of the included studies were published in 1992 or later. As described in section 2, we did not set a lower boundary for the year of publication in the search process, yet the time frame of identified studies reflects also the time frame of the evolution and maturation of software architecture area. The significant increase of publications in software architecture evolution area, especially during the last two years, indicates that, as more and more systems become legacy over the years, the crucial role of software architecture evolution is being recognized. The recent boost in research also reflects that the ability to evolve software rapidly and reliably has become a major challenge and research focus for software engineering.

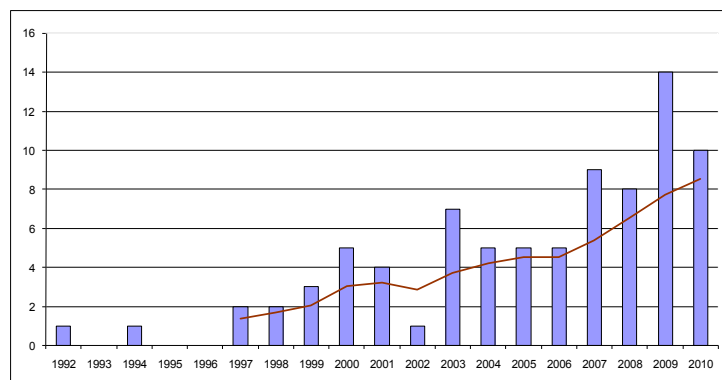


Figure 3. Number of Papers by Year of Publication

### 3.4 Active Research Communities

In terms of the active research communities within the area of software architecture evolution and software evolvability, we look at the affiliation details<sup>5</sup> of the identified set of studies. Table 6 summarizes the active research communities (with at least two publications within software

<sup>5</sup> Please note that during the search process of relevant studies, we did not use any information on authors or research centers for identifying studies because the result of identified studies would be otherwise limited and biased.

architecture evolution) along with the corresponding number of contributed studies. Overall, the set of studies are dominated by Software Engineering Institute (SEI)/Carnegie Mellon University, Vrije University, and University of Groningen.

Table 6. Active Research Communities within Software Architecture Evolution

Affiliations	Number of Contributed Studies
Software Engineering Institute, Carnegie Mellon University, USA	11
Vrije University, the Netherlands	8
University of Groningen, the Netherlands	7
University of Texas, USA	5
Blekinge Institute of Technology/ University of Karlskrona/Ronneby, Sweden	5
University Rey Juan Carlos, Spain	4
Swinburne University of Technology, Australia	4
National ICT Australia, Australia	3
University of Limerick, Ireland	3
University of New South Wales, Australia	3
University of Waterloo, Canada	3
Imperial College of Science, England	2
Mälardalen University, Sweden	2
ABB Corporate Research, Sweden	2
Nokia Research Center, Finland	2
Technical University Ilmenau, Germany	2
Texas Christian University, USA	2
University College London, England	2

## 4. Results

As described in section 2, during the data synthesis phase, we examined the identified studies based on their similarities in terms of research topics and contents in order to categorize the included studies of architecture evolution and software evolvability. Besides classifying the included studies, we also examined analysis methodology for collected data in each study, and application context for each approach described in the studies. After examining the research topics addressed in each study, we identified, from the included studies, five main categories of themes, two of which are further refined into sub-categories to group primary studies that share similar characteristics in terms of specific research focus, research concepts and contexts. The categories and sub-categories are:

- 1) **Quality considerations during software architecture design:** This category focuses on how software quality can be introduced and explicitly considered during software architecture design phase.
  - a. Quality attribute requirement focused (S8, S10, S13, S25, S26, S27, S79)
  - b. Quality attribute scenario focused (S24, S30)
  - c. Influencing factor focused (S1, S29, S31, S38, S42, S80)
- 2) **Architectural quality evaluation:** This category focuses on the subsequent iteration when the architecture starts to take form, with emphasis on architectural quality

evaluation methods that help elicit and refine additional quality attribute requirements and scenarios.

- a. Experience based (S14, S34, S37, S50, S73)
  - b. Scenario based (S11, S33, S47, S48, S53, S54, S62)
  - c. Metric based (S5, S15, S16, S28, S55, S56, S57, S67, S71, S75)
- 3) **Economic valuation:** This category focuses on consideration of cost, effort, value and alignment with business goals, when determining an appropriate degree of architectural flexibility. (S4, S6, S7, S9, S18, S23, S35, S46, S64, S66, S72)
  - 4) **Architectural knowledge management:** This category focuses on how architecture documentation can be enriched through utilizing different information sources to capture architectural knowledge for quality attributes and their rationale. (S2, S3, S12, S19, S20, S21, S22, S36, S40, S43, S44, S45, S52, S68, S70, S77, S78, S82)
  - 5) **Modeling techniques:** This category focuses on modeling traceability and visualizing corresponding impact of the evolution of software architecture artifacts. (S17, S32, S39, S41, S49, S51, S58, S59, S60, S61, S63, S65, S69, S74, S76, S81)

Figure 4 illustrates these categories of themes and their corresponding sub-categories along with an overview of distribution of studies.

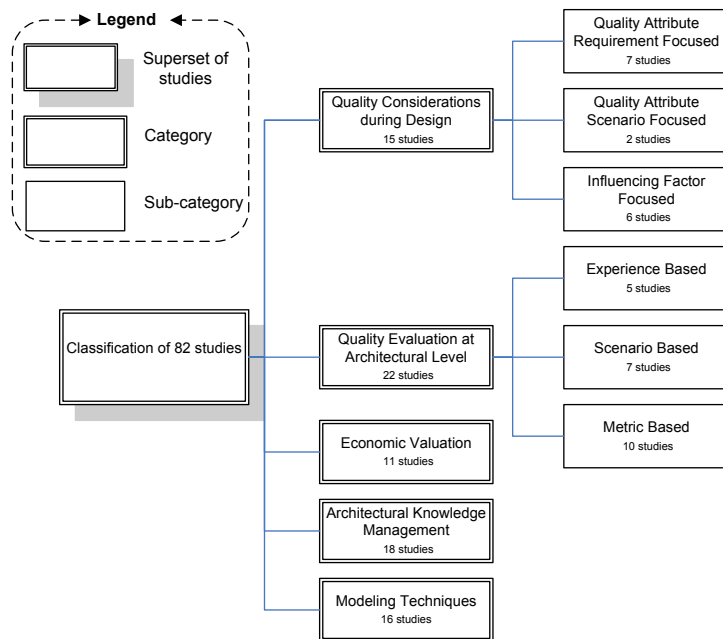


Figure 4. Classification of Included Studies

The categories and their corresponding sub-categories will be further detailed in the following subsections. For each category of theme, we describe the category and related studies, along with their relevance to software evolvability. An analysis of the studies is discussed and summarized in tables. Each table includes the following items: (i) the main focus and application context of each approach, including issues such as constraints and limitations; (ii) the techniques adopted in each approach; and (iii) research validation environment.

## 4.1 Quality Considerations during Software Architecture Design

This category includes studies that focus on *how software quality can be introduced and explicitly considered during software architecture design phase*. These studies help identify key quality attributes and constraints early, usually before the software architecture starts to take form. Based on their focus, the studies are further classified into three sub-categories: (i) quality attribute requirement-focused; (ii) quality attribute scenario-focused; and (iii) influencing factor-focused.

### 4.1.1. Quality Attribute Requirement-Focused

The studies in this sub-category perceive quality attribute requirements as the main focus in the software architecture design phase, and consider each design decision based on its implications on the prioritized quality attributes.

- *Attribute-Driven Design (ADD)* [S8] is a recursive top-down method for architects to hierarchically decompose a system and define software architecture by applying architectural tactics and patterns. It is applied after the requirement analysis phase in the lifecycle to accomplish a software system's coarse-grained high-level conceptual architecture. The driving forces in the design process include functional requirements, quality attribute requirements and design constraints that are well-formed and prioritized by stakeholders. ADD centralizes around prioritized requirements. The secondary requirements are fulfilled within the constraints of the most important ones.
- *Quality Attribute Oriented Software Architecture Design (QASAR)* [S10, S13] describes a *software design method* that explicitly considers quality attributes during the design process. The method consists of three key phases, i.e. functionality-based architecture design, architecture assessment and architecture transformation. The design process starts with an application architectural design based on the functional requirements without explicitly addressing quality requirements. This design is then evaluated with respect to quality requirements qualitatively or quantitatively to achieve an estimated value for each quality attribute. Depending on whether or not the estimated value satisfies the requirement specification, an architecture transformation might be required for quality attribute optimization.
- *Architectural prototyping* [S25] is an another technique to design software architectures by using executable code to investigate architectural quality attributes that are related to stakeholders' concerns with respect to a system under its development.
- *Non-Functional Requirement (NFR) framework* [S27] is a process-oriented and qualitative decomposition approach for eliciting and analyzing non-functional requirements. It systematically takes into consideration the conflicts and synergies between NFRs in order to develop an evolvable architecture. The operation of the framework is visualized through soft-goal interdependency graphs in which quality requirements are treated as soft-goals to be achieved. High-level soft goals are refined into more specific sub-goals. To satisfy each sub-goal, design decisions are reinforced with a design rationale. One limitation of the framework is that it treats all NFRs as soft goals that are to be "*satisfied*", i.e. not absolutely achieved but within acceptable limits. This might lead to ambiguity in requirement specifications when

there is a need to characterize and quantify hard goals, e.g. requirements in hard real time systems. One example of *using NFR framework along with design patterns* for developing adaptable software architecture is described in [S26]. This approach takes into consideration particular characteristics of software system domain, and refines quality requirements into architectural concepts and alternatives, that are subsequently satisfied with design patterns.

- *Adaptability Evaluation Method (AEM)* [S79] is an integral part of the Quality-driven Architecture Design and quality Analysis (QADA) methodology [21] with specialization in the adaptability aspect. AEM defines adaptability goals through capturing the adaptability requirements that will be subsequently considered in the architecture design. In this study, guidelines on how to model adaptability in architectural models are provided. The approach is used to qualitatively/quantitatively analyze candidate architectures to ensure that adaptability requirements are met before system implementation.

**Relevance to software evolvability**

Except [S26] and [S79], the other approaches address software quality attributes in general, and can be tailored to address evolvability by focusing on evolvability subcharacteristics and by considering the impacts of a design decision on these subcharacteristics. Both [S26] and [S79] explicitly address adaptability, though the definitions of adaptability differ. In [S79], adaptability is regarded as a qualitative property of software architecture’s maintainability (which is a superset of flexibility, integrability, testability and modifiability) and includes runtime requirements of the software system as well as adaptation to changes in stakeholders’ requirements. In [S26] adaptability is perceived to be heavily dependent on a particular software development project’s scope and nature. This approach only focuses on few design patterns that enhance adaptability of real-time software systems and does not address the multifaceted evolvability perspective of long-lived software systems.

A summary of quality attribute requirement-focused approaches is given in Table 7, describing the main focus and application context of each approach, along with issues such as constraints and limitations; the techniques adopted in each approach as well as research validation environment. Although these approaches focus on quality attribute requirements, they differ from each other. The NFR framework considers quality attributes as soft goals, i.e. there is no clear-cut definition and criteria as to whether they are satisfied or not. This is in contrast with ADD in which quality attribute requirements are well-formed and prioritized. Besides quality attributes, ADD also considers functional requirements as primary drivers in the design process. This is in contrast with QASAR method, which conceives functional requirements as the primary driver for creating application architectural design, whereas quality attributes are treated as secondary drivers and are not considered a driving force in the first phase of the architecture development.

Table 7. Summary of Quality Attribute Requirement-Focused Approaches

<b>Study</b>	<b>Focus and Application Context</b>	<b>Included Technique</b>	<b>Validation</b>
S8	Focus on prioritized requirements, i.e. functional requirements, quality attribute requirements and design constraints. Assist architects in making design decisions based on their effects on quality attributes	Recursive top-down design	Validated in various domains

S10, S13	The design process separates architectural design based on functional requirements and quality requirement optimization. An iterative design process to optimize architecture.	Several optimization techniques are used, e.g. scenarios, simulations, mathematical modeling	Validated in the embedded systems domain
S25	Investigate architectural qualities and stakeholders' concerns by using executable code.	Experimental technique	Validated in various domains
S26	Require clarifications of the notion of adaptability in order to refine adaptability requirements. Particular domain characteristics are considered.	NFR – soft goal interdependency graph. Design patterns. Qualitative tradeoff analysis of impact	Illustrated by a home appliance control system
S27	Treat non-functional requirements as soft goals. Considers each design decision based on its effects on the quality attributes. Does not provide support to explicitly perform tradeoff analysis between competing design decisions.	NFR framework with soft goal interdependency graph	Validated in various domains
S79	Identify stakeholders and their concerns. Qualitative and/or quantitative analysis of adaptability depending on the knowledge of components' behavior.	Strategic Dependency Model (SDM). Objective reasoning for qualitative analysis	Validated in an industrial case study in wireless environment controlling system

#### 4.1.2. Quality Attribute Scenario-Focused

The studies in this sub-category focus on mapping architectural quality goals into concrete scenarios to characterize stakeholders' concerns throughout the software architecture design phase.

- *Software architecture analysis* method [S24] defines several steps in the software design process: (i) architectural quality goals are expressed through scenarios to characterize the generic quality goals in concern; (ii) mechanisms are tailored to realize the scenarios within the design constraints; and (iii) analytic models are instantiated by scenarios that represent quality attributes of interest or potential risk areas in architecture. The constitution of the analytic models is an iterative process due to the ever-changing architectural requirements and design constraints. As the system evolves, the analytic models can be used to assess the impact of architectural changes and monitor how architectural evolution affects its capability to support predicted modifications.
- *Active Reviews for Intermediate Designs (ARID)* [S30] is a scenario-based assessment method for evaluating intermediate design or parts of an architecture for early feedback. It is a lightweight method that can be used to judge if the design of a partial architecture is appropriate for its intended purpose before the development of the complete architecture. The stakeholders involved in ARID brainstorm and prioritize scenarios that represent foremost problems the design is expected to handle, in order to assess the suitability of a design approach and discover discrepancies.

#### **Relevance to software evolvability**

Applying the software architecture analysis approach in [S24] would require quite a number of evolvability scenarios to address and cover evolvability subcharacteristics. A more important limitation is that while scenarios are anticipated events in the system life-time, evolvability by nature concerns also unanticipated events which calls for some more general type of analysis. The approach in [S30] focuses more on scenarios that represent foremost problems the design is

expected to handle rather than considering a system’s long-term evolvability aspect. Therefore, this approach needs to be complemented with more explicit consideration of scenarios that would cover evolvability concern and subcharacteristics.

A summary of quality attribute scenario-focused approaches is given in Table 8. All approaches utilize quality attribute scenarios though with distinct purposes; the scenarios in [S24] are used for concretizing architectural quality goals, whereas the scenarios in [S30] are used to identify most important functions, issues and problems that are embedded in intermediate design.

Table 8. Summary of Quality Attribute Scenario-Focused Approaches

Study	Focus and Application Context	Included Technique	Validation
S24	Architectural quality goals are mapped into scenarios, mechanisms that realize the scenarios, and analytic models that measure the results	Scenarios. Analytic models	Validated with example scenarios from two real-life software systems
S30	Judge the appropriateness of a partial architecture for its intended purpose during architecture design	Active design review. Scenarios. Stakeholder-centric	Validated in various domains

#### 4.1.3. Influencing Factor-Focused

The studies in this sub-category focus on, early in the design phase, managing factors that are architecturally significant, and constraints that have influence on the design process, along with inter-dependencies among these factors and constraints that would affect the choice of design decisions.

- *ArchDesigner* [S1] is a quantitative quality-driven design approach for architectural design process. The approach evaluates stakeholders’ quality preferences and design alternatives. Meanwhile, a software architecture design problem is considered as a global optimization problem due to the inter-dependencies among different design decisions that need to be maintained, as well as global constraints that influence the selection of any design alternative, e.g. project constraints. Optimization techniques are thus used to determine an optimal combination of design alternatives. The influencing factors that are systematically managed are factors that influence the design process, including conflicting stakeholders’ quality goals, various design decisions, design alternatives and inter-dependencies, architectural concerns and project constraints.
- *Business goal elicitation* [S29] empowers architects to articulate business goals among stakeholders early in the lifecycle, and is used as prelude to architecture evaluation.
- *Architecture-Based Component composition/Decision-oriented Design (ABC/DD)* [S31] accomplishes architecture design from design decision perspective, by eliciting architecturally significant design issues and exploiting corresponding solutions for these issues.
- *Incorporation of changeability* within a system architecture is a concept introduced in [S38]. It proposes four aspects that have influence on changeability: (i) flexibility that characterizes a system’s ability to be changed easily; (ii) agility that characterizes a system’s ability to be changed rapidly; (iii) robustness that characterizes a system’s ability to be insensitive towards changing environment; and (iv) adaptability that characterizes a system’s ability to adapt itself



to changing environments. These four aspects can be implemented depending on the needed type and extent of changeability.

- *Global analysis* [S42] provides a systematic way to identify and describe architecturally significant factors in the design phase to be able to develop strategies to accommodate these factors, and reflect future concerns early for making design decisions. The influencing factors are classified into three categories: (i) *organizational* factors that constrain design choices; (ii) *technological* factors, such as choices of hardware, software, architecture technology, and standards; (iii) *product* factors that cover a product's functional features and qualities. All these factors interact with each other. They need to be aggregated and prioritized. New factors that may arise during design need to be considered as well. Afterwards, issues that are influenced by these factors are identified, and specific strategies that address the issues are developed to reduce the impact of various factors.
- *Design constraint-oriented approach* [S80] enhances understanding of architectural decision making by treating design constraints, i.e. external forces that restrict an architect's choice of solution space, as central constructs of architecture.

#### **Relevance to software evolvability**

The *ArchDesigner* approach in [S1] addresses quality attributes in general and can be tailored to assess stakeholders' preferences on evolvability subcharacteristics, and determine preferences of design alternatives based on the weighting scores of evolvability subcharacteristics. The *Business goal elicitation* approach in [S29] is systematic in identifying primary business drivers for performing an evolvability analysis. Both [S31] and [S80] provide respectively, a qualitative indication on how the choice of a design decision/design constraints, would affect evolvability. The concept in [S38] does not cover the other evolvability subcharacteristics except changeability. The qualities addressed in [S42] emphasize more on operational-related qualities rather than development-oriented quality attributes of a software system such as evolvability. However, identifying organizational factors and technical constraints is relevant to determining strategies in architecting for evolvability.

A summary of influencing factor-focused approaches is given in Table 9. All these approaches focus on identifying influencing factors, though with varying perspectives of influencing factors and presence of strengths and weakness. For instance, *Global analysis* uncovers architecturally significant factors including quality attributes in the early lifecycle of architecture design. There is a clear traceability between influencing factors and derived strategies. But the reasoning about quality consequences of each design decision is not sufficiently supported. This weakness is complemented by [S1], which performs value score computation on stakeholders' preferences on quality attributes and weighting design alternatives' consequences on quality attributes. The *Business goal elicitation* approach focuses on an organization's business goals and ties them to quality attribute requirements, whereas *ABC/DD* [S31] focuses on architecturally significant design issues, and [S80] on design constraints.

Table 9. Summary of Influencing Factor-Focused Approaches

Study	Focus and Application Context	Included Technique	Validation
S1	Quantitatively determine the optimal design alternative that best satisfy stakeholders' quality goals and project constraints. Observed limitations in judgment uncertainties and judgment consistency.	Interviews. Optimization techniques. Analytic Hierarchy Process (AHP)	Validated as a post-mortem analysis of a production software system for information analysts.
S29	Capture business goals early in the lifecycle.	Business goal scenarios	Validated in Boeing system
S31	Provides an iterative process to implement the architecture design. Issue relationship at different levels is not handled.	Decision abstraction. Issue decomposition principle.	Validated in two large scale projects
S38	Changeability incorporates four aspects, i.e. robustness, flexibility, agility and adaptability.	Theoretical reasoning	Illustrated by examples from varying industries
S42	Identify architecturally significant factors early in the design phase and develop strategies.	Global analysis	Validated in various domains
S80	Identify design constraints and analyze their impact on architecture.	Design constraint properties.	Validated in industrial systems

## 4.2 Quality Evaluation at Software Architecture Level

An architecture assessment is triggered by various business goals [19], such as evaluating and improving architecture and its qualitative attributes, identifying architectural drift and erosion, identifying risks related to a particular architecture. From evolution perspective, architecture evaluation is a preventive activity to delay architectural decay and to limit the effect of software aging [31]. The studies in this category focus on *quality evaluation at the architecture level when the software architecture starts to take form after the initial design phase*. Based on their focus, the studies are further classified into three sub-categories: experience-based, scenario-based and metric-based evaluation.

### 4.2.1 Experience-based

Experience-based architecture evaluation means that evaluations are based on previous experiences and domain knowledge of developers or consultants [1]. The studies in this sub-category focus on extracting experiences of stakeholders and making use of their tacit knowledge. The evaluation process is mostly based on subjective factors such as intuition and experience.

- *Lightweight sanity check for implemented architectures (LiSCIA)* method [S14] focuses on maintainability and reveals potential problems as a software system evolves. This method detects erosion by interviewing system developers using five categories of questions: current grouping of units in modules and future modules, decomposition of functionality over modules, module size, module dependencies, and technologies. The limitations of LiSCIA are: (i) it depends heavily on the evaluator's opinion; (ii) it only aims to discover potential risks related to maintainability; (iii) the use of only a single viewpoint (module viewtype) sets a limit to covering all potential risks.
- *Knowledge-based assessment approach* [S34] evaluates the evolutionary path of software architecture during its lifecycle based on the knowledge of the stakeholders involved in the software development organizations. The extraction of knowledge and factual evidence of claims requires representativeness and completeness in the selection of stakeholders. The drivers for using this method include lack of formal and complete architecture documentation, wide scope of assessment, large number of stakeholders, and geographical distribution of

development teams. The outcomes of the assessment are current architecture overview, main issues found, and optionally, recommendations for their resolutions.

- The concept of identifying *causes for changes* and *strategies to cope with changes* during a system's lifecycle is described in [S37]. This concept is based on analyzing projects that are already finished and extracting experiences on the most frequent changes in terms of sources of stimuli and cost of each change.
- *Attribute-Based Architectural Style (ABAS)* [S50] explicitly associates architectural styles with reasoning frameworks based on quality-attribute-specific models for particular quality attributes. ABAS consists of four parts: (i) *problem description* that explains the problem being solved by the software structure; (ii) *stimuli and response* that correspond to the condition affecting the system and measurement of the activity as a result of the stimuli; (iii) *architectural styles* that are descriptions of component interaction patterns; and (iv) *analysis* that constitutes a quality-attribute-specific model for reasoning about the behavior of interacting components in the pattern. A specific attribute-based architectural style is accompanied with a set of questions. These questions and answers to the questions are accumulated as a knowledge base that can be exploited during architectural reviews.
- *Decision support method* [S73] quantitatively measures stakeholders' views of the benefits and liabilities of software architecture candidates and relevant quality attributes. The method is used to understand and choose optimal candidate architecture among software architecture alternatives. Although the primary data collection is comprised of subjective judgments, influenced by the knowledge, experiences and opinions of stakeholders, the data collection of stakeholders' subjective opinions is quantifiable. Thus, any disagreements between the participating stakeholders can be highlighted for further discussions.

### **Relevance to software evolvability**

The *LiSCIA* approach [S14] focuses only on maintainability from module viewpoint with respect to dependencies in order to detect erosions. Although the *Knowledge-based assessment approach* [S34] addresses evolvability, there is no description on the authors' perception of evolvability. Lacking explicit consideration of the multifaceted feature of software evolvability, this approach might miss some key aspects that are critical for software evolution. Heavily dependent on stakeholders' subjective interpretation of quality attributes, the *Decision support method* [S73] faces a similar issue. The *ABAS* reasoning framework [S50] is based on quality-attribute-specific models for particular quality attributes. It does not take into account the tradeoff relationships among quality attributes. Though, in order to determine potential evolutionary paths of an architecture, the preferences and tradeoffs among evolvability subcharacteristics must be considered.

A summary of experience-based quality evaluation approaches is given in Table 10. These approaches differ from each other mainly in two aspects:

- (i) *Method for data collection*; In [S50, S73], the method for primary data collection is a questionnaire that individual participating domain expert fills out. One possible drawback with a questionnaire is that ambiguous questions might lead to problematic interpretations by participants

due to their differing experiences. For instance, [S73] purposely planned to provide less detailed descriptions of architecture candidates in order to provide more room for participants, though with the risk of problematic interpretations of the architecture candidates and relevant quality attributes by participants. As a countermeasure, interviews as in [S14], [S34] and [S37], can be used to complement questionnaires, clarify questions for respondents, capture additional information to the answers from questionnaires, as well as unexpected responses.

(ii) *Delivered output of quality evaluation.* The *Knowledge-based assessment* approach in [S34] focuses on identification of key issues that are critical for software evolution. Resolutions to these issues are optional, whereas the *Decision support method* [S73] aims to reach a shared view of resolutions in terms of the choice of architecture candidate by discussing about identified disagreements among stakeholders. An accumulated knowledge base for future exploitation is the main output for [S37] and [S50].

Table 10. Summary of Experience-Based Quality Evaluation Approaches

Study	Focus and Application Context	Included Technique	Validation
S14	Detect erosion when it has happened.	Interview. List of questions and actions.	Validated in various industrial domains.
S34	Knowledge-based assessment. Stakeholder-centric: rely on experiences of stakeholders. Implicit iteration in the process. Requires well-focused assessment scope and careful selection of stakeholders.	Semi-structured interviews.	Validated in an industrial mobile terminal product family
S37	Five strategies to cope with change. Prevention and front-loading strategy needs to be complemented with building changeability into system architecture.	Questioning through questionnaire and interviews.	An exploratory case study in telecommunication domain
S50	Associate a qualitative or quantitative reasoning framework with an architectural style	Questionnaire/checklist.	Validated in various domains
S73	A quantified decision support method that creates increased joint understanding on the choice of software architecture candidates and quality attributes. Risk in problematic interpretation of questionnaire questions, architecture candidates and quality attributes. Rely on experiences of stakeholders. Require sufficient participants to achieve reliable measures.	Questionnaire. Analytic Hierarchy Process (AHP). Discussion meetings.	Validated as an industrial experiment on a software system in automatic guided vehicles system domain with experienced practitioners.

#### 4.2.2 Scenario-based

Scenario-based architecture evaluation means that quality attributes are evaluated by creating scenario profiles for a concrete description of a quality requirement [22]. The studies in this sub-category use scenarios to avoid terminological ambiguities and conflicting interpretation of quality attributes.

- *Software Architecture Analysis Method (SAAM)* [S47, S30] was originally created for evaluating modifiability of software architecture although it has been used for other quality attributes as well, such as portability and extensibility. The primary inputs to the evaluation include system architecture descriptions and scenarios that describe a stakeholder's interaction

with the system. Based on these, SAAM establishes a mapping between architecture and the scenarios that represent possible future changes to the system. This mapping provides indications of potential future complexity parts in the software and estimated amount of work related to changes.

- *Architecture Tradeoff Analysis Method (ATAM)* [S48, S30] evolves from SAAM and evaluates multiple quality attributes for understanding the tradeoffs inherent in the software architecture. It is used to uncover implicit requirements, and reveal how well an architecture satisfies particular quality attributes. It provides insight into how these quality attributes interact with each other, by exposing risks, non-risks, sensitivity points and tradeoff points in the software architecture.
- *Holistic Product Line Architecture Assessment (HoPLAA)* method [S62] is an extension to ATAM for assessing product line architecture. This method is performed in two stages to identify risks at two architecture levels: core architecture evaluation, and individual product architecture evaluation. During core architecture evaluation, evolvability points are identified and evolvability guidelines are defined. The notion of evolvability points designates a sensitivity point or a tradeoff point that contains at least one variation point. The identification of evolvability points ensures that quality attributes at individual product architecture level do not preclude core architecture quality attributes. Evolvability guidelines are used to inform designers about potential conflicts, and guide them to make appropriate design decisions in subsequent product architecture design phase.
- *Architecture Level Modifiability Analysis (ALMA)* [S11, S53, S54] analyzes modifiability based on scenarios that capture future events a system needs to adapt to in its lifecycle. The method consists of five steps: setting analysis goal, software architecture description, change scenarios elicitation, change scenarios evaluation, and interpretation of the results. Depending on the goal of analysis, the output from an ALMA evaluation varies among: (i) maintenance prediction to estimate required effort for system modification to accommodate future changes; (ii) architecture comparison for optimal candidate architecture; and (iii) risk assessment to expose the boundaries of software architecture by explicitly considering environment and using complex change scenarios that the system shows inability to adapt to.
- *A scenario-based assessment method* [S33] *evaluates evolvability of software product line architecture* towards forthcoming requirements. The method consists of three phases: (i) scenario collection, classification and prioritization; (ii) architecture evaluation based on the chosen scenarios; and (iii) assessment result compilation. The output includes potential flaws and evolutionary path of the software architecture.

### **Relevance to software evolvability**

Both SAAM and ATAM would require quite a number of evolvability scenarios to address all evolvability subcharacteristics. A more important limitation is that while scenarios are anticipated events in a system life-time, evolvability by nature concerns unanticipated events which calls for some more general type of analysis. The approaches in [S11], [S53] and [S54] do not cover the other evolvability subcharacteristics except changeability, and thus need to be complemented with other methods to address all evolvability subcharacteristics. In [S33] evolvability of software

product line architecture is evaluated towards forthcoming requirements without providing a definition of evolvability. Moreover, this approach provides little guidance in scenario selection, which makes it difficult to develop scenarios that would cover all software evolvability subcharacteristics. The approach in [S62] assesses only product line architecture and does not focus on the evolution of other types of architecture.

A summary of scenario-based quality evaluation approaches is given in Table 11. These approaches exhibit a variety of characteristics. In [S47], the scenarios proposed by stakeholders determine the quality attributes for analysis, whereas in [S48], the quality attributes for analysis are synthesized through explicitly considering both business and technical perspectives. *ALMA* focuses only on modifiability and has distinguished analysis goals which determine the choice of change scenarios and techniques used in the analysis process. For instance, for risk assessment, complex scenarios, guided interview and system environment modeling techniques are used; for maintenance cost prediction, scenarios that are likely to occur during the operational lifecycle are used; for architecture comparison purpose, scenarios that are handled differently in architecture alternatives are used. One limitation of the method is that the evaluation of change scenario with respect to its ripple effects on other components relies much on architects' experiences.

Table 11. Summary of Scenario-Based Quality Evaluation Approaches

Study	Focus and Application Context	Included Technique	Validation
S11, S53, S54	Focus on modifiability. Pursue maintenance prediction, risk assessment and software architecture comparison. Goal of the analysis determines techniques for the analysis process, e.g. scenario elicitation technique and scenario evaluation technique.	Brainstormed change scenarios. Scenario classification scheme. Scenario weight estimation. Interviews.	Validated in various domains
S33	Lightweight analysis method tuned to software product line architecture. Iterative process with focus on evolvability. Stakeholder-centric. Little guidance to scenario selection and ranking process.	Scenarios. Interviews. Brainstorming session.	Validated in Nokia multimedia software domain
S47	Qualitative assessment. Iterative scenario development. Provide few explicit techniques for the analysis process and relies much on the assessor's experiences.	Brainstormed scenarios. Voting procedure for scenario prioritization. Rank by assigning weights.	Validated in various domains
S48	Qualitative tradeoff analysis. Identify architectural risks in light of business goals. Consider multiple quality attributes and identify tradeoffs between quality attributes. Explicitly consider both business and technical perspectives. Assess consequences of architectural decisions in light of quality attributes.	Utility tree. Brainstormed scenarios. Voting procedure for scenario prioritization.	Validated in various domains
S62	Focus on risks and quality attributes for both common product line architecture and individual product architecture. Identification of evolvability points and evolvability guidelines. Need further validation and refinement through applying to real life product line architectures.	Utility tree. Brainstormed scenarios. Voting procedure for scenario prioritization.	Demonstrated as an industrial trial

### 4.2.3 Metric-based

The studies in this sub-category assess quality impact qualitatively or quantitatively through specific quality metrics.

- Besides *implementation change logs* [S67] and *computation of metrics using the number of modules* in a software system [S56], another set of metrics is based on *software life span and software size* [S75]. Software evolution can also be quantitatively analyzed by using *evolution ratio* which is the amount of evolution in terms of software size, and *evolution speed* which is an indicator of an organization's capability for software system's evolution [S5].
- A framework of *process-oriented metrics for software evolvability* [S71] develops intuitively architectural evolvability metrics, and traces the metrics back to the evolvability requirements based on the NFR framework [8]. Similarly, *process-oriented metric for software architecture adaptability* [S28] analyzes the degree of adaptability through intuitive decomposition of goals and intuitive scoring of goal-satisfying level of software architecture. As the method depends much on intuition and expert expertise, [S57] proposes a *quantitative metric-based approach to evaluate software architecture adaptability*. This approach supports decision-making in choosing architecture candidates that meet stakeholders' adaptability goals that are expressed in scenario profiles. The impact of each scenario profile is measured through *IOSA (impact on the software architecture)* and *ADSA (adaptability degree of software architecture)*.
- A *software evolvability model* is outlined in [S15], in which subcharacteristics of software evolvability and corresponding measuring attributes are identified. The subcharacteristics that are of primary importance for long-lived software-intensive systems' evolvability include *analyzability, architectural integrity, changeability, extensibility, portability, testability* and *domain-specific attributes*. Measuring attributes for each subcharacteristic are identified as well. The idea with this model is to further refine the identified subcharacteristics to the extent when it is possible to quantify them and/or make appropriate reasoning about the quality of the attributes. Based on this evolvability model, [S16] presents an *evolvability analysis method* which ensures that the implications of potential improvement strategies and evolution path of a software architecture are analyzed with respect to the evolvability subcharacteristics.
- A *tradeoff analysis method of architecture using architecture analysis and design language* [S55] acquires quantitative values from an architecture model by establishing and measuring metrics of quality attributes.

#### **Relevance to software evolvability**

Both [S15] and [S16] explicitly address software evolvability, and provides a base and check point for evolvability evaluation and improvement. Both [S28] and [S57] explicitly address software adaptability, i.e. "*the system's ability to make adaptation, which involves environment change detection, system change recognition and system change enactment*" [S28]. The focus of these studies is around changeability subcharacteristic and does not cover other evolvability subcharacteristics, e.g. analyzability, testability and architectural integrity. Although [S71] focuses on software evolvability, it does not provide any precise definition of evolvability. Instead, the

study advocates that the definition and decomposition of evolvability is determined by the domain. This is in conformance to the domain-specific attributes defined in evolvability subcharacteristics.

A summary of metric-based quality evaluation approaches is given in Table 12.

Table 12. Summary of Metric-Based Quality Evaluation Approaches

Study	Focus and Application Context	Included Technique	Validation
S5	Base on evolution ratio and evolution speed.	Metrics.	Empirical study in mobile phone software systems
S15, S16	Refine evolvability into seven subcharacteristics that are measured through measuring attributes.	Subcharacteristics and measuring attributes.	Validated in industrial automation domain
S28	Process-oriented qualitative framework for representing and reasoning about adaptability. Depend much on intuition and expert expertise which leads to uncertainty.	NFR framework.	Academic experiment
S55	Quantitatively measure quality attributes for analyzing architecture.	Quality attributes and metrics. Architecture analysis and design language.	Validated in automotive industry
S56	Computation of metrics using the number of modules.	Metrics.	Validated in a financial transaction system
S57	Quantitatively measure and evaluate adaptability through adaptability scenario profile and impact analysis.	Scenario profile. Metrics.	Theoretical reasoning
S67	Base on implementation change logs. More applicable for evaluating maintenance activities instead of evolvability.	Metrics.	Validated with the evolution of kernel of a mainframe operating system
S71	Process-oriented, capture design rationale. Even experienced software engineers need training to do evolvability-related NFR decompositions.	NFR framework.	Two industrial-scale systems, with more than 50000 lines of code
S75	Base on software life span and software size.	Metrics.	Theoretical reasoning

### 4.3 Economic Valuation in Determining Level of Uncertainty

The uncertainties in software architecture evolution arise from, to a certain extent, understanding how architectural decisions map onto quality attribute responses in terms of costs and benefits. The studies in this category *cope with uncertainty in determining an appropriate degree of architectural flexibility and balance with economic valuation to mitigate risks in investment.*

- One way to address economic valuation is to estimate the required effort for system modification to accommodate future changes. For instance, *maintenance cost prediction* [S9] 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. The underlying prediction model is based on the estimated change volume and productivity ratios. *Maintenance effort prediction* during architecture design is another method [S4], which takes requirements, domain knowledge and general software engineering knowledge as input to prescribes application architecture, and quantifies maintenance effort by classifying weighted scenarios in terms of complexity.
- Instead of only focusing on cost/effort analysis, *Cost Benefit Analysis Method (CBAM)* [S46] is an architecture-centric economic modeling approach that can address long-term benefits of a change along with its implications in complete product lifecycle. This method quantifies



design decisions in terms of cost and benefits analysis, and prioritizes changes to architecture based on perceived difficulty and utility. Another *cost-benefit framework for making architectural decisions* is proposed in [S23]. This approach correlates the change in developer effort to the change in coupling by analyzing a categorized set of modifications to specific software components both before and after an architectural refactoring. *Architecture Improvement Workshop (AIW)*<sup>6</sup> is another method for taking economic considerations – cost, benefits, and uncertainty, into account by setting values on architectural decisions in relation to quality attributes.

- Software architecture decisions carry economic value in form of real options [2, 30]. Options offer flexibility and allow architectural evolution over time [S6, S35]. A model for *predicting the stability of software architectures using real options* is exploited in [S6], which advocates that the flexibility of an architecture to endure changes in stakeholders' requirements and environment has a value in predicting stability of the software architecture. To maximize the lifetime value of a software architecture, [S35] *incorporates the concept of architecture options into design* in order to exploit quantitatively an optimal degree of design flexibility. In [S64] the authors hypothesize that architectural patterns carry economic value in the form of real options, and propose to consider *cost, value and alignment with business goals to support architectural evolution*. This approach guides the selection of design patterns, elicitation of architecturally significant requirements, and valuation of architecture in terms of design decisions with multiple quality-attribute viewpoints. The approach in [S7] provides insights into *architectural flexibility and investment decisions* related to the evolution of software systems by examining probable changes along with their added value, such as accumulated savings through enduring the change without violating architectural integrity, supporting future growth, and capability of responding to competitive forces and changing market conditions. The approach in [S72] *uses design structure matrices to model designs and real options technique to value designs*.
- Given particular schedule constraints, an appropriate degree of *architectural flexibility* [S66] can be determined *through four strategic elements*: feature prioritization, schedule range estimation, core capability determination and architecture flexibility determination. The intention is to mitigate the risk of violating schedule, cost and quality constraints.
- Based on several key parameters that have perceived value to a system's stakeholders, [S18] proposes a conceptual approach to *quantify a system's life cycle value* to facilitate adaptability to changes in circumstances and stakeholder preferences.

### **Relevance to software evolvability**

Software evolvability concerns both business and technical perspectives as the choice of design decisions when architecting for evolvability needs to be balanced with economic valuation to mitigate risks. Several studies focus on a single quality attribute, e.g. stability in [S6, S7],

---

<sup>6</sup> There is no publication on this topic yet. Therefore, it is not included in the systematic review. Details on this topic can be found at <http://www.sei.cmu.edu/architecture/consulting/aiw/index.cfm> (visited on 7th of September, 2010)

flexibility in [S35, S66] and modularity in [S72], and do not explicit consider the multifaceted aspects of evolvability. Both [S46] and [S64] covers multiple quality attributes. However, *CBAM* relies on the output from *ATAM* which might not be an appropriate method for extracting scenarios to cover all evolvability subcharacteristics (as explained in section 4.2.2). The approach in [S64] focuses only on the value of architectural patterns for quality attributes that are of interest to stakeholders, and fails to take into account the preferences and tradeoffs among evolvability subcharacteristics.

A summary of economic valuation approaches is given in Table 13. All these approaches consider at least one of the following, i.e. cost, effort, value and alignment with business goals, when determining an appropriate degree of architectural flexibility.

Table 13. Summary of Economic Valuation Approaches

Study	Focus and Application Context	Included Technique	Validation
S4	Predict maintenance efforts at architectural level.	Growth scenario profile. Scenario classification with respect to complexity.	Illustrated with a web content extraction application architecture
S6	View stability as a strategic architectural quality that adds values in form of growth options. Value flexibility.	Real options theory	Theoretical reasoning
S7	Provide insight into architectural stability and software evolution investment decisions.	Real options theory	Academic experiment of a refactoring case study
S9	Augment architecture description with size estimates. Prediction of maintenance efforts. Dependency on domain experts and architects. Lack of validation the representativeness of a maintenance profile.	Change scenarios. Prediction model.	Exemplified in the medical equipment domain
S18	Quantify lifecycle value of enduring systems.	Surveys and interaction with stakeholders. Market surveys and user group assessment.	Exemplified with a cellular telephone system
S23	Correlate change in developer effort to the change in coupling. Compute predicted savings in effort.	Compute average change in coupling and effort	Validated in a marketing services company
S35	Static and dynamic evaluation of architecture flexibility.	Real options theory. Metrics. Optimization techniques.	Illustrated with quantitative examples
S46	Analyze cost and benefits of architectural strategies. Sensitivity to uncertainty in cost and benefit values. Rely on ATAM to identify architecture strategies.	Quality attributes scores. Benefit and cost quantification.	Validated in various domains
S64	Consider cost, value and alignment with business goals when exploiting option values of an architectural pattern.	Real options theory	Theoretical reasoning
S66	Model-based approach to assist in determining an appropriate degree of architectural flexibility within constraints. Need further calibration and validation of architecture flexibility determination model.	Expert judgment. Parametric cost modeling.	Academic experiment in a full text system
S72	Modularity in design creates value in the form of real options. Model design and value the design.	Data structure matrices. Real options theory.	Illustrated with Parnas' KWIC example

## 4.4 Architectural Knowledge Management

The studies in this category focus on *utilizing various information sources to capture architectural knowledge*, which is comprised of architecture design, design decisions, assumptions, context, and other factors that together shape a software architecture. In spite of the exhibited properties of large software systems [6], e.g. software complexity, inevitable changes of software systems and invisibility of software structure representation, architectural integrity needs to be maintained. An explicit representation of architectural knowledge is therefore necessary for evolving systems and assessing future evolutionary capabilities of a system [15].

- Apart from using change scenarios and change cases to model variability and describe future evolutionary capabilities, it is also useful to explicitly *model invariability assumptions*, i.e. things that are assumed will not change [S52]. Assumptions are design decisions and rationale that are made out of personal experience and background, domain knowledge, budget constraints and available expertise. There are three types of assumptions: *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. The discovery and recovery of architectural knowledge in terms of assumptions help assess the evolutionary capabilities of system architecture. These assumptions can also 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 traceability between architecture evolution and early-made assumptions would supplement design decisions to confront uncertainties when predicting future user requirement changes. A relevant method is *Recovering Architectural Assumptions Method (RAAM)* [S68] that makes assumptions explicit by recapitulating historical information of software system evolution.
- To assess architectural design erosion [32], an *architecture assessment model measures the extent of deviation in terms of functional and structural divergence* [S12]. In order to track software evolution, the loss of system functionality and architectural structure are represented using functional and structural erosion indicators respectively, indicating whether changes that are incorporated into a system would violate integrity of architectural design.
- As architectural constraints influence the quality of architectural design process and improvement of software quality, the concept of *classifying architectural constraints* [S40] is used to generalize architectural styles and patterns.
- *Documenting architectural design decisions (ADD)* is another approach to maintain architectural artifacts in order to evolve software in a controlled way without compromising software integrity [4]. [S77] reports on practitioners' perception of the value, usage and documentation of design rationale, and argues for the need of tool support for capturing and using design rationale to avoid knowledge vaporization and dependency on domain experts. In line with this reclamation, several tools have been developed [S2, S3, S20, S21, S22, S36, S43, S44, and S45] for sharing design decisions along with rationale. [S19, S70, S78] provide comparative studies of these architecture knowledge management tools. [S70] suggests another tool for visualization of design decisions and rationale, in order to overcome the

deficiencies in the existing tools, e.g. visualization support for dependency relationship between ADDs, support for collaborate decision-making, and rationale visualization support.

- *Mining patterns to systematically extract and document architecturally significant information* [S82] improves architecture evaluation activities for pattern-oriented systems. General scenarios and architectural tactics are extracted from software patterns, and are used as input to architecture evaluation, and vice versa, the architecture evaluation results provide input to pattern validation.

### **Relevance to software evolvability**

The studies in this sub-category focus on capturing architectural knowledge, and therefore are useful in improving architectural integrity which is one of the evolvability subcharacteristics.

A summary of architectural knowledge management approaches is given in Table 14. To achieve a good understanding of decisions that sustain an architecture, [S52, S68] capture assumptions that architectural decisions are often based on. [S20, S36, S44, S45] focus specifically on capturing and managing design decisions and rationale for functional requirements, whereas [S2, S3, S21, S22] pay more attention to capturing quality attributes knowledge, i.e. design decisions and rationale for quality attributes. [S20, S22] further distinguishes from other studies with its explicit emphasis on architecture views. [S44, S45] consider software architecture as a composition of a set of architectural design decisions. [S44] focuses on recovering architectural design decision for the purpose of reverse engineering, whereas [S45] maintains the relationships between design decisions for the purpose of forward engineering. Both approaches have a similar architectural design decision model, though [S45] extends the decision model by combining it with a meta-model that is comprised of an architectural model, a requirement model and a composition model. This allows architects to document architectural design decisions with traceability to related requirements and part of the implementations. However, the evolution perspective is not explicitly addressed in [S45]. Besides codifying architectural knowledge that concerns an architecture, [S36] distinguishes from the above mentioned studies with one supplementary feature, i.e. architectural knowledge sharing using personalization techniques.

Table 14. Summary of Architectural Knowledge Management Approaches

<b>Study</b>	<b>Focus and Application Context</b>	<b>Included Technique</b>	<b>Validation</b>
S2, S3	Capture design decisions and rationale for quality attributes, and provide knowledge repository. Explicitly augment quality attribute utility tree with design decisions. No support on diagrammatic modeling of design decisions. Need to be integrated with appropriate requirement management tool to avoid work duplication.	Open source groupware platform, i.e. Hipergate. Data model.	Validated as an industrial trial in architecture evaluation process
S12	Objectively measure the extent of architectural deviation in the system. Might have limitations in handling large scale legacy system.	Abstract architectural model representation. Architectural erosion measures.	Validated in a sample university registration system
S20	Capture design decisions and rationale for functional requirements. Less attention is paid for recording quality attribute knowledge.	Argumentation representation. Argument ontology.	Validated in a set of experiments
S21,	Provide support for capturing design decisions for quality attributes and their rationale.	Mandatory and optional attributes.	Validated in a virtual reality system

S22	Describe and document explicitly tacit knowledge. Selection of mandatory and optional attributes for capturing design decisions.		
S36	Integrated functionality supports architects in decision-making process. Architectural knowledge sharing. Less attention is paid for recording quality attribute knowledge.	Architectural knowledge sharing portal.	Validated as an experiment in a software development organization
S40	Capture high level architectural design knowledge. Cover only a subset usage of architectural styles.	Taxonomy based on ANSI/IEEE 1471 standard.	Theoretical meta study based on empirical research results
S43	Add formal architectural knowledge (AK) through annotating the existing documented AK sources based on a formal meta model.	Domain model. Formal meta-model. Plug-ins.	Validated through a large industrial example
S44	Iterative process of recovering architectural design decisions. High dependency on architects for the recovery process.	Tool support.	Validated in an academic experiment
S45	Tool support at the later stages within design to bind architectural decisions, models and system implementation. Less attention is paid for recording quality attribute knowledge. Not explicitly address design decision evolution perspective.	Architectural description language integrated with Java.	Validated in an academic experiment
S52, S68	Utilize different information sources to capture assumptions in order to assess the architecture's evolutionary capabilities. Evolutionary aspects of assumptions are not addressed.	Source code access. Historical information, e.g. development process statistics. Interviews. Documentation.	Validated in an e-commerce software product
S70	Support explicit rationale visualization of an architectural design decision.	Argumentation-based approach.	Not validated yet
S77	Empirical investigation of use and documentation of design rationale.	Surveys.	A survey of practitioners
S19, S78	Comparative study of architectural knowledge tool support.	Comparison framework of 10 criteria.	Not applicable
S82	Improve software architecture design and evaluation through mining patterns. Initial work on improving architecture evaluation activities for pattern oriented systems.	Scenarios. Tactics.	Validated in an academic demonstration by using EJB architecture usage patterns

## 4.5 Modeling Techniques

Due to the fact that all artefacts produced and used during the entire software lifecycle are subject to change, the studies in this category mainly focus on *modeling artifacts to support software architecture evolution*.

- *Modeling traceability links between requirements, features, architectural elements and implementation* is described in [S17] to improve evolvability. A formal definition of indicators that concern evolvability deficiency and corresponding resolution actions is provided as well.
- To assess software architectures for evolution and reuse, a *framework in modeling relevant information and architectural views* [S58] is proposed for reengineering, analyzing, and comparing software architectures. The types of information for traceability modeling include: (i) *stakeholder information* that describes stakeholders' objectives, and provide boundaries for analysis; (ii) *architecture information* such as design principles or architectural objectives; (iii) *quality information*; and (iv) *scenarios* that 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 assess the architecture's support for potential enhancements. In this way, sensitivity points of a system are revealed. A *lightweight traceability management* concept [S51] proposes to customize traceability by scoping the traces that need to be maintained to only activities stakeholders must carry out.

- The approach in [S63] focuses on *managing quality properties* during the whole lifecycle of model-driven development. Besides using model and quality-driven architecture design/evaluation, this approach is extended with knowledge engineering, and involves three main phases: modeling reusable quality requirements, representing quality in architectural models, and model-based quality evaluation on whether the desired quality goals are met in models and code.
- *Using architectural tactics to embody non-functional requirements (NFRs) into software architecture* is described in [S49]. These tactics are reusable architectural building blocks that provide generic solutions to quality attribute issues. The tactics along with their relationships are represented in *Feature models*, whereas the structure and behavior of tactics are described using the *Role-Based Modeling Language* (RBML) [13]. Another tactic-based modeling is *tactic-based non-functional requirement (NFR) modeling* approach [S59] incorporates NFRs into software analysis and design phase. Based on a classification framework of tactics types, the approach focuses on tactics of NFRs rather than the NFRs themselves, and manages tradeoffs among competing NFRs by considering prioritization and impact of tactics on NFRs.
- *A concern-driven software development approach* [S61] supports developers in understanding and evolving software systems. A concern is a concept that relates a group of software fragments. The approach consists of three main elements: (i) a fine-grained *concern model* that associates each concern to the set of artifacts that implement the concern; (ii) visualization of concerns at both code level and architectural level; and (iii) automated support in maintaining concern model over time.
- The approach in [S32] uses *ontology to visualize architectural design decisions* by means of scenarios such as quality attribute tradeoff analysis, impact analysis and if-then scenarios. Another *ontological approach for architectural style modeling* [S65] is based on description logic. Instead of using ontology to model architectural style, [S76] proposes to evolve software architecture by using graph transformations to provide a formal specification of evolution patterns.
- *Modeling an evolvable system by building a wrapper-system* [S60] coordinates three stages of iteration: capturing system behavior, updating system state, and applying new changes. By using a clustering algorithm, [S69] identifies software layers for understanding and evolution of object-oriented software systems. To allow architects to precisely express and reason about architecture evolution with the goal of choosing an optimal evolution path for an architecture, [S39] focuses on (i) *evolution path*, which is a first-class entity for representation and analysis; and (ii) *evolution style*, which defines a family of domain-specific architecture evolution paths that share common properties and satisfy a common set of constraints.

- *Modeling change impact* [S41] between software architecture and its related source code is performed by using (i) Architectural Software Component Model (ASCM) which represents software architecture descriptions; (ii) typology of change operations; (iii) formalized change propagation mechanism; and (iv) defined change propagation process.
- To address evolution of system requirements and software architecture, *quality-driven software reengineering model* [S74] adopts NFR Framework [8] and the concept of soft goals to support modeling of design rationale with soft-goal interdependency graphs.
- The approach in [S81] focuses on business rules, which represent an important source of requirement changes due to their high impact on software and business process. Business rules are considered as an integral part of system evolution, and are specified in *Business Rule Model*, which is then related to meta-model level of software design elements through a *Link Model*. Modeling business rules improves requirement traceability in software design, and helps in localizing impacts of changing business rules.

### **Relevance to software evolvability**

The modeling-techniques help improve architecture evolution by modeling the relationships among inter-dependent software artefacts, which if not handled with care, would introduce inconsistencies and lead to evolvability degradation in the long run.

A summary of modeling techniques is given in Table 15.

Table 15. Summary of Modeling Techniques

<b>Study</b>	<b>Focus and Application Context</b>	<b>Included Technique</b>	<b>Validation</b>
S17	Model relations between requirements, features, architectural elements and implementation for evaluating and improving evolvability.	Traceability modeling. Features.	Validated in an industrial IT infrastructure domain
S32	Model architectural design decisions using ontology-driven visualization.	Ontology instances.	Validated in a product audit organization
S39	Model evolution paths with the goal of choosing an optimal path to achieve business objectives. Characterize recurring patterns as a set of evolution styles.	Utility-theoretic approach.	Theoretical.
S41	Model change impact on the structure of software architecture.	Rule-based approach.	Implementation based on Eclipse Development Environment.
S49	Model architectural tactics in feature models, and define semantics for these tactics.	Feature modeling. Role-based meta-modeling language	Demonstrated with a stock trading system.
S51	Scope for a minimum set of links to model traceability.	Traceability path	Illustrated with examples, i.e. product line engineering and process management
S58	Model various types of information, i.e. stakeholder, architecture, quality and scenarios. Risk level indication through estimating the required effort (low, medium, or high) to make the changes. Analysis is based on stakeholder objectives. Require upfront modeling and compilation of various stakeholders' perspectives.	Traceability modeling. Scenarios. Architecture views. Quality function deployment.	Empirical study in a large scale telecommunication switching system
S59	Model tactics as opposed to focusing on NFRs themselves.	NFR framework. Qualitative and quantitative analysis.	Illustrated with a case study of Automatic Teller Machine (ATM) application

S60	Construct a wrapper system which generates feedback data, and detects the need for evolutionary changes.	Object-process modeling.	Validated by analyzing system usage activity logs and update request history of projects
S61	Model concerns and map them towards software artifacts.	Concern model.	Three small evaluations assessing different aspects
S63	Model quality requirements to create quality attribute ontology and requirements models. Quality driven model selection from architectural knowledge base. Model based quality evaluation (qualitative and quantitative)	Ontology. Model-driven engineering. Domain specific modeling. Scenarios. Quantitative measuring techniques, prediction methods, measurement based methods.	Validated in a secure middleware project
S65	Conceptual modeling of architectural styles.	Ontology. Description logic.	Illustrated with an example.
S69	Identify software layers for the understanding and evolution of existing object-oriented software systems.	Clustering algorithm.	Empirical investigation
S74	Model NFR requirements to guide software transformation. Not explicitly address the estimation of transformation impact.	NFR framework. Soft-goal inter-dependency graphs. Design patterns.	Validated with two medium-size software systems (less than 9 KLOC)
S76	Use sequences of architectural restructurings to specify architecture evolution.	Graph transformations.	Validated with an Internet shop application
S81	Model business rules as an integral part of a software system evolution. Improved traceability between requirements and design.	Model. Typology.	Validated in a healthcare information system

## 5. Discussions

The identified categories of themes provide an overview of software architecture evolvability research as well as a basis for discovering possibilities for improvement in research and practice. The following sections discuss potential impact on research and practice, as well as validity threats in this review.

### 5.1 Impacts on Research and Practice

This systematic review has a number of implications for research and practice.

#### 5.1.1. Technology Maturation

This systematic review provides us a perspective of where the field of architecture evolution and software evolvability stands today. To get better understanding of the development of the field, we examine the maturity phase of the approaches described in the included studies by mapping them against Redwine-Riddle model [27], which identifies six typical phases for technology maturation, typically taking 15-20 years for a technology to enter widespread use.

- *Basic research*: investigation of ideas and concepts, and articulation of research questions;
- *Concept formulation*: informal circulation of ideas and convergence on a compatible set of ideas;



- *Development and extension*: exploration of preliminary use of the technology, clarification of underlying ideas, and generalization of the approach;
- *Internal enhancement and exploration*: extension of the general approach to other domains, usage of the technology to solve real problems, and stabilization of the technology;
- *External enhancement and exploration*: involvement of a broader group outside the development group to show substantial evidence of value and applicability of the technology;
- *Popularization*: appearance of production-quality, supported versions and commercialization of the technology.

All the three authors of this article reviewed the 82 studies and cataloged independently the maturation classification of the technology presented in each study. When there were any discrepancies in the judgment on maturation level of any studies, discussions were then initiated in order to reach an agreement. Figure 5 summarizes the classification results (number of studies indicated in parenthesis for each maturation phase and maturation distribution in percentage) according to the technology maturation model.

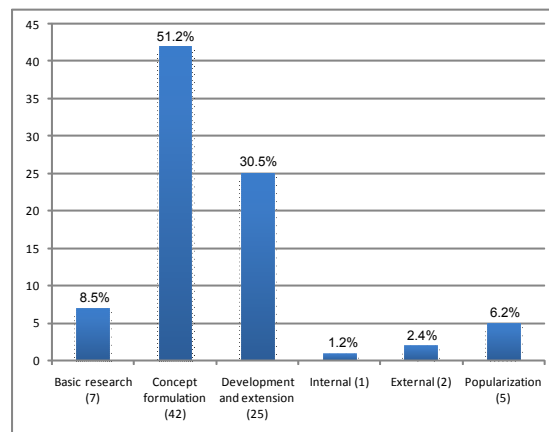


Figure 5. Technology Maturation Classification of Included Studies

We can see from the classification result that a large majority of the 82 studies belong to early maturity stages; almost 60% of studies belong to early stages (basic research and concept formulation), while around 30% of studies come to the development and extension phase. This implies that most methods and tools are still not widely established in industrial practices, indicating that the value and applicability of many novel research ideas still need to be further extended on industrial projects of various scales and in different industrial domains.

### 5.1.2 Theoretical Foundation to Software Architecture Evolution

The 82 studies concern two main aspects: (i) development of new, or modification of existing approaches to support architecture evolution and software evolvability; (ii) evaluation of the effect of applying an approach. To get a good understanding of how the approaches have been assessed, we examine the included studies by looking into the empirical method they use, e.g. theoretical

reasoning, single-case validation in industry, etc. A distribution of the studies per validation status is shown in Table 16.

Table 16. Study Distribution per Empirical Method used(**Categ. 1** Quality consideration during design; **Categ. 2** Quality evaluation at architectural level; **Categ. 3** Economic valuation; **Categ. 4** Architectural knowledge management; **Categ. 5** Modeling techniques)

<b>Empirical Method</b>	<b>Categ. 1</b>	<b>Categ. 2</b>	<b>Categ. 3</b>	<b>Categ. 4</b>	<b>Categ. 5</b>	<b>Number</b>	<b>%</b>
Single-case in Industry	3	12	5	6	8	34	41.5
Single-case in Academia	1	1	2	6	5	15	18.3
Multiple-case	9	7	1	0	1	18	21.9
Theoretical Reasoning	1	0	3	4	2	10	12.2
Survey	1	2	0	2	0	5	6.1
<b>Total</b>	15	22	11	18	16	<b>82</b>	<b>100%</b>

About one-fifth (21.9%) of the studies have extended their approaches for solving industrial problems in multiple domains. Two out of the five surveys were conducted on practitioners in companies. Most of the case studies are single-case, with 34 studies done in projects in industry and 15 studies in academic settings. Moreover, 8 studies are on theoretical level, indicating also the challenge in collecting empirical data due to the complex and longitudinal nature of software evolution. As we see from the table, 63.4% (i.e. 41.5% + 21.9%) of the studies include industrial case studies, and 71.7% (i.e. 41.5% + 18.3% + 21.9%) include case studies. This large percentage of case studies implies: (i) software evolution research studies real-world phenomena, and the knowledge is acquired on the basis of case studies rather than deductive logic, mathematics, or generalized knowledge, as generalizing the results from case studies to settings beyond the studied organizations is a challenge; (ii) architecture evolution and software evolvability is still in an early phase of research, which is unlike mature sciences where theory building is a principle method of accumulating knowledge to be used in different settings; (iii) software evolution research area is by its nature, due to its complexity, is more difficult to be explained by theoretical principles than by practical experiences; thus, a theoretical foundation with practical value for software evolution is necessary.

### *5.1.3. Combining Approaches to Address Multifaceted Perspectives of Software Evolvability*

Each of the approaches identified in the review has its specific focus and context that it is appropriate for. For instance, the Attribute Driven Design (ADD) [S8] assists in making design decisions based on their effects on quality attributes. The input to its commencement depends on some analysis results from other methods, e.g. Quality Attribute Workshop (QAW)<sup>7</sup> which helps in understanding the problem by eliciting quality attribute requirements in the form of quality attribute scenarios. Moreover, ADD uses prioritization of quality attributes when the choice of

<sup>7</sup> There is no publication on this topic in the electronic databases. Details on this topic can be found at <http://www.sei.cmu.edu/architecture/consulting/qaw/index.cfm> (visited on 22<sup>nd</sup> of September, 2010)

architectural patterns and tactics cannot support all the desired quality attributes. In this context, ADD depends on some kind of architecture evaluation method, e.g. ATAM, in order to analyze how each design alternative would influence the tradeoffs among all desired quality attributes. Therefore, considering the architectural design activities in the software lifecycle, ADD needs to be complemented with approaches that support elicitation of quality requirements as well as approaches that support reasoning about choice of design alternatives.

Another example is related to scenario-based analysis methods. Most scenario-based software architecture analysis methods have the strength of being able to concretize driving quality attribute requirements, but they also have a weakness of being optimistic in change scenario elicitation due to the unpredictable nature of changes as well as stakeholders' short horizon in foreseeing future changes [17]. Therefore, some architectural knowledge management approaches can be used to complement scenario-based methods and address this weakness through explicit representation of invariabilities to provide additional what-if scenarios. Economic valuation methods can also be used to complement with details on business consequences of architectural decisions. Another weakness of most scenario-based analysis methods is their lack of a more fine-grained analysis [S58], although most of these approaches are effective for high-level evaluation of an architecture. Modeling techniques can thus be used to complement with traceability information and visualization of impact analysis.

We see an emerging initiative in research community to combine appropriate techniques for software architecture evolution [9, 26]. As evolvability needs to be addressed over the complete software lifecycle, it is necessary to combine appropriate approaches to manage this multifaceted attribute [S15].

#### ***5.1.4 Tailoring Relevant Approaches for Individual Development Contexts***

For practitioners, this review presents a wide spectrum of approaches that analyze and improve software evolvability from specific perspectives. As described in section 4, each approach identified in the review has its specific application context that it is appropriate for, such as the required input for commencement when using an approach, the phase in the software lifecycle when an approach is suitable, scope of analysis and output, etc. Thus, this review can be used by practitioners as a source in searching for relevant approaches. We suggest that the main consideration for practitioners is to carefully examine the context and characteristics of their own project, and compare with the application context and constraints of a certain approach before adopting and tailoring the approach into their own software development.

### **5.2 Validity Threats**

The main threats to validity in this systematic review are bias in our selection of the studies to be included, and data extraction. To be able to identify relevant studies and ensure that the process of selection was unbiased, we developed a research protocol that defined research questions, inclusion and exclusion criteria, and search strategy. The research protocol was reviewed by several senior researchers for feedback and was modified based on their comments. However, due

to our choice of search terms, there is still a risk that we might have missed some relevant studies, especially in cases when some software engineering keywords are not standardized and clearly defined, such as definitions for various quality attributes. We dealt with this threat by making sure that all the researchers participating in this review had the same definition in case of unclear terms, though in some cases it was hard to know how the authors of the reviewed papers defined for example adaptability or evolvability. Another threat is related to old publications that are probably only in paper format, and have not been scanned and stored in the searched electronic databases. To further ensure the unbiased selection of articles, we performed a multi-step selection process 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. To ensure correctness in data extraction, we defined a data extraction form to obtain consistent extraction of relevant information for answering the research questions. In addition, we performed quality assessment on relevant studies to ensure that the identified findings and implications came from a credible basis.

## 6. Conclusions

As business and technology evolve and software becomes more complex, software development is increasingly faced with not only how to create new software systems of the desired quality attributes, but also, following the initial development, how to evolve the systems in their operationally changing contexts. Given that in most cases it is not desirable to develop everything from scratch [24], researchers are constantly challenged to come up with approaches to effectively support the evolution of software systems. The main objective of this review is to obtain a holistic view of the existing studies in analyzing and achieving software evolvability at architectural level. We have identified 82 primary studies, covering a spectrum of approaches with specific perspective or focus on a particular architecture-centric activity in software lifecycle. These approaches vary in terminology, descriptions, artifacts and involved activities, yet beyond these differences, we find approaches that share a lot in common, e.g. focus, goal and application context. We extract these commonalities and summarize the studies into five main categories of themes:

1. *Quality considerations during design.* The approaches in this category are further refined into three sub-categories: quality attribute requirement focused, quality attribute scenario focused, and influencing factor focused. Most of the techniques that support quality considerations during software architecture design help identify key quality attribute requirements early in the software design phase. Most studies address quality attributes in general and not evolvability in particular.
2. *Architectural quality evaluation.* In the subsequent iteration when an architecture starts to take form, architectural quality evaluations help elicit and refine additional quality attribute requirements and scenarios. The approaches in this category are further refined into three sub-categories: experienced-based, scenario-based and metric-based. A reflection on how these studies are related to software evolvability is that most studies focus on particular quality attributes such as adaptability, and do not cover the wide

spectrum of evolvability subcharacteristics. Few studies explicitly address software evolvability. Even if the term *evolvability* is used in some studies, there is a lack of precise definition or explanation of authors' perception on software evolvability.

3. *Economic valuation.* Economic valuation approaches provide more details on architectural decisions' business consequences, and assist development teams in choosing among architectural options. Most studies focus on a single quality attribute, e.g. stability, flexibility or modularity, and may exhibit a drawback in architectural design decision-making process when multiple evolvability subcharacteristics are involved, requiring explicit management of preferences and tradeoffs among evolvability subcharacteristics.
4. *Architectural knowledge management.* Architectural knowledge management approaches improve architectural integrity by enriching architecture documentation with architectural knowledge captured from different information sources.
5. *Modeling techniques.* Modeling techniques add value by modeling software artefacts along with their traceability, and visualizing corresponding impact of the evolution of software architecture artifacts. They do not explicitly focus on evolvability in particular, but they help control and improve software architecture evolution by modeling the relationships among inter-dependent software artefacts.

This systematic review might have implications for both research and practitioners. For researchers, an analysis of the primary studies indicates that there is a need to establish a foundation of theories with practical value to software architecture evolution. It is also necessary to address the multifaceted perspectives of software evolvability through combining appropriate approaches to complement each other as each approach has its specific focus and context that it is appropriate for in a software lifecycle. For practitioners, they can use this review as a source in searching for relevant approaches before adopting and tailoring them by examining the context and characteristics of their own software development, and comparing with the application context of relevant approaches.

In future we can expect more research work in this area – in addition to case studies we could expect more basic foundation research and standardization of designing, and assessing evolvability, probably enriched by different tools.

## Appendix: Studies Included in the Review

- [S1] T. Al-Naeem, J. Gorton, M. Ali Babar, F. Rabhi, B. Benatallah, A quality-driven systematic approach for architecting distributed software applications, International Conference on Software Engineering (ICSE), 2005.
- [S2] M. Ali Babar, I. Gorton, A tool for managing software architecture knowledge, International Conference on Software Engineering Workshop on Sharing and Reusing architectural Knowledge-Architecture, Rationale, and Design Intent, 2007.
- [S3] M. Ali Babar, I. Gorton, R. Jeffery, Capturing and using software architecture knowledge for architecture-based software development, International Conference on Quality Software (QSIC), pp. 169-176, 2005.
- [S4] S. Anwar, M. Ramzan, A. Rauf, A. Ali Shahid, Software maintenance prediction using weighted scenarios: an architecture perspective, International Conference on Information Science and Applications (ICISA), 2010.
- [S5] M. Aoyama, Continuous and discontinuous software evolution: aspects of software evolution across multiple product lines, International Conference on Software Engineering Workshop on Principles of Software Evolution, pp. 87-90, 2001.
- [S6] R. Bahsoon, W. Emmerich, ArchOptions: a real options-based model for predicting the stability of software architectures, International Conference on Software Engineering Workshop on Economic-Driven Software Engineering Research, 2003.
- [S7] R. Bahsoon, W. Emmerich, Evaluating architectural stability with real options theory, International Conference on Software Maintenance, pp. 443-447, 2004.
- [S8] L. Bass, P. Clements, R. Kazman, Software architecture in practice, ISBN 0321154959, Addison-Wesley Professional, 2003.
- [S9] P. Bengtsson, J. Bosch, Architecture level prediction of software maintenance, European Conference on Software Maintenance and Reengineering (CSMR), pp. 139-147, 1999.
- [S10] P. Bengtsson, J. Bosch, Scenario-based software architecture reengineering, International Conference on Software Reuse, pp. 308-317, 1998.
- [S11] P. Bengtsson, N. Lassing, J. Bosch, H. van Vliet, Architecture-level modifiability analysis (ALMA), Journal of Systems and Software, vol. 69, pp. 129-147, 2004.
- [S12] S. Bhattacharya, D. E. Perry, Architecture assessment model for system evolution, Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007.
- [S13] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach, Addison-Wesley Professional, ISBN 0-201-67494-7, 2000.
- [S14] E. Bouwers, A. van Deursen, A lightweight sanity check for implemented architectures, IEEE Software, vol. 27, 2010.
- [S15] H. P. Breivold, I. Crnkovic, P. J. Eriksson, Analyzing Software Evolvability, Annual IEEE International Computer Software and Applications Conference (COMPSAC), 2008.
- [S16] H. P. Breivold, I. Crnkovic, R. Land, M. Larsson, Analyzing software evolvability of an industrial automation control system: a case study, International Conference on Software Engineering Advances (ICSEA), 2008.
- [S17] R. Brcina, M. Riebisch, Architecting for evolvability by means of traceability and features, International Conference on Automated Software Engineering (ASE) Workshops, pp. 72-81, 2008.
- [S18] T. R. Browning, E. C. Honour, Measuring the life-cycle value of enduring systems, Journal of Systems Engineering, vol. 11, 2008.
- [S19] W. Bu, A. Tang, J. Han, An analysis of decision-centric architectural design approaches, International Conference on Software Engineering Workshop on Sharing and Reusing Architectural Knowledge (SHARK), 2009.
- [S20] J. E. Burge, D. C. Brown, Software Engineering Using RATIONALE, Journal of Systems & Software, vol. 81, pp. 395-413, 2008.
- [S21] R. Capilla, F. Nava, J. C. Dueñas, Modeling and documenting the evolution of architectural design decisions, International Conference on Software Engineering Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent, 2007.
- [S22] R. Capilla, F. Nava, S. Pérez, J. C. Dueñas, A web-based tool for managing architectural design decisions, ACM SIGSOFT Software Engineering Notes, vol. 31, 2006.
- [S23] J. Carriere, R. Kazman, I. Ozkaya, A cost-benefit framework for making architectural decisions in a business context, International Conference on Software Engineering, 2010.
- [S24] S. J. Carriere, R. Kazman, S. G. Woods, Assessing and maintaining architectural quality, European Conference on Software Maintenance and Reengineering, pp. 22-30, 1999.
- [S25] H. B. Christensen, K. M. Hansen, An empirical investigation of architectural prototyping, Journal of Systems and Software, vol. 83, pp. 133-142, 2010.
- [S26] L. Chung, K. Cooper, A. Yi, Developing adaptable software architectures using design patterns: an NFR approach, Computer Standards & Interfaces, vol. 25, pp. 253-260, 2003.
- [S27] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, Non-functional Requirements in Software Engineering, ISBN 978-0-7923-8666-7, Springer, 2000.
- [S28] L. Chung, N. Subramanian, Process-oriented metrics for software architecture adaptability, IEEE International Symposium on Requirements Engineering, pp. 310-311, 2001.

- [S29] P. Clements, L. Bass, Business goals as architectural knowledge, International Conference on Software Engineering Workshop on Sharing and Reusing Architectural Knowledge (SHARK), 2010.
- [S30] P. Clements, R. Kazman, M. Klein, Evaluating Software Architectures: Methods and Case Studies, ISBN 0-201-70482-x, Addison-Wesley, 2006.
- [S31] X. Cui, Y. Sun, S. Xiao, H. Mei, Architecture design for the large-scale software-intensive systems: a decision-oriented approach and the experience, International Conference on Engineering of Complex Computer Systems (ICECCS), 2009.
- [S32] R. C. de Boer, P. Lago, A. Telea, H. van Vliet, Ontology-driven visualization of architectural design decisions, Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA), 2009.
- [S33] C. Del Rosso, Continuous evolution through software architecture evaluation: A case study, Journal of Software Maintenance and Evolution, vol. 18, pp. 351-383, 2006.
- [S34] C. Del Rosso, A. Maccari, Assessing the architectonics of large, software-intensive systems using a knowledge-based approach, Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007.
- [S35] A. Engel, T. R. Browning, Designing systems for adaptability by means of architecture options, Journal of Systems Engineering, vol. 11, 2008.
- [S36] R. Farenhorst, R. Izaks, P. Lago, H. van Vliet, A Just-In-Time Architectural Knowledge Sharing Portal, Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 125-134, 2008.
- [S37] E. Fricke, B. Gebhard, H. Negele, E. Igenbergs, Coping with changes: Causes, findings, and strategies, Journal of Systems Engineering, vol. 3, pp. 169-179, 2000.
- [S38] E. Fricke, A. P. Schulz, Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle, Journal of Systems Engineering, vol. 8, 2005.
- [S39] D. Garlan, J. M. Barnes, B. Schmerl, O. Celiku, Evolution styles: foundations and tool support for software architecture evolution, Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 2009.
- [S40] S. Giesecke, W. Hasselbring, M. Riebisch, Classifying architectural constraints as a basis for software quality assessment, Advanced Engineering Informatics, vol. 21, pp. 169-179, 2007.
- [S41] M. O. Hassan, L. Deruelle, H. Basson, A knowledge-based system for change impact analysis on software architecture, International Conference on Research Challenges in Information Science, 2010.
- [S42] C. Hofmeister, R. Nord, D. Soni, Applied Software Architecture: A Practical Guide for Software Designers, ISBN 0201325713, Addison-Wesley Professional, 2000.
- [S43] A. Jansen, P. Avgeriou, J. S. van der Ven, Enriching software architecture documentation, Journal of Systems and Software, vol. 82, pp. 1232-1248, 2009.
- [S44] A. Jansen, J. Bosch, P. Avgeriou, Documenting after the fact: Recovering architectural design decisions, Journal of Systems & Software, vol. 81, pp. 536-557, 2008.
- [S45] A. Jansen, J. Van der Ven, P. Avgeriou, D. K. Hammer, Tool support for architectural decisions, Working IEEE/IFIP Conference on Software Architecture (WICSA) 2007.
- [S46] R. Kazman, J. Asundi, M. Klein, Quantifying the costs and benefits of architectural decisions, International Conference on Software Engineering 2001.
- [S47] R. Kazman, L. Bass, G. Abowd, M. Webb, SAAM: a method for analyzing the properties of software architectures, International Conference on Software Engineering, pp. 81-90, 1994.
- [S48] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 68-78, 1998.
- [S49] S. Kim, D. K. Kim, L. Lu, S. Park, Quality-driven architecture development using architectural tactics, Journal of Systems and Software, vol. 82, pp. 1211-1231, 2009.
- [S50] M. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, H. Lipson, Attribute-based architecture styles, Working IEEE/IFIP Conference on Software Architecture (WICSA) 1999.
- [S51] P. Lago, H. Muccini, H. van Vliet, A scoped approach to traceability management, Journal of Systems and Software, vol. 82, pp. 168-182, 2009.
- [S52] P. Lago, H. van Vliet, Explicit assumptions enrich architectural models, International Conference on Software Engineering, pp. 206-214, 2005.
- [S53] N. Lassing, P. Bengtsson, H. van Vliet, J. Bosch, Experiences with ALMA: Architecture-Level Modifiability Analysis, Journal of Systems and Software, vol. 61, pp. 47-57, 2002.
- [S54] N. Lassing, D. Rijsenbrij, H. van Vliet, How well can we predict changes at architecture design time?, Journal of Systems and Software, vol. 65, pp. 141-153, 2003.
- [S55] J. Lee, D. H. Lee, Quantitative tradeoff analysis of software architecture using the architecture analysis and design language, ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009.
- [S56] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Pery, W. M. Turski, Metrics and laws of software evolution-the nineties view, 4th International Symposium on Software Metrics, 1997.
- [S57] X. Liu, Q. Wang, Study on application of a quantitative evaluation approach for software architecture adaptability, International Conference on Quality Software (QSIC), pp. 265-272, 2005.
- [S58] C. H. Lung, S. Bot, K. Kalaichelvan, R. Kazman, An approach to software architecture analysis for evolution and reusability, Conference of the Centre for Advanced Studies on Collaborative Research, IBM Center for Advanced Studies Conference, 1997.

- [S59] T. Marew, J.S. Lee, D. H. Bae, Tactics based approach for integrating non-functional requirements in object-oriented analysis and design, *Journal of Systems and Software*, vol. 82, pp. 1642-1656, 2009.
- [S60] A. Mubin, D. Ray, R. Rahman, Architecting an evolvable system by iterative object-process modeling, *World Congress on Computer Science and Information Engineering (CSIE)*, 2009.
- [S61] E. C. Nistor, A. van der Hoek, Explicit concern-driven development with ArchEvol, *IEEE/ACM International Conference on Advanced Software Engineering*, 2009.
- [S62] F. G. Olumofin, V. B. Mistic, A holistic architecture assessment method for software product lines, *Information and Software Technology*, vol. 49, pp. 309-323, 2007.
- [S63] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, P. Aho, Knowledge based quality-driven architecture design and evaluation, *Journal of Information and Software Technology*, vol 52, pp.577-601, 2010.
- [S64] I. Ozkaya, R. Kazman, M. Klein, Quality-Attribute Based Economic Valuation of Architectural Patterns, *International Workshop on the Economics of Software and Computation*, 2007.
- [S65] C. Pahl, S. Giesecke, W. Hasselbring, Ontology-based modeling of architectural styles, *Journal of Information and Software Technology*, vol. 51, pp. 1739-1749, 2009.
- [S66] D. Port, L. Huang. Strategic architectural flexibility, *International Conference on Software Maintenance (ICSM)*, pp. 389-396, 2003.
- [S67] J. F. Ramil, M. M. Lehman, Metrics of software evolution as effort predictors-a case study, *International Conference on Software Maintenance (ICSM)*, pp. 163-172, 2000.
- [S68] R. Roeller, P. Lago, H. van Vliet, Recovering architectural assumptions, *Journal of Systems and Software*, vol. 79, pp. 552-573, 2006.
- [S69] G. Scanniello, A. D'Amico, C. D'Amico, T. D'Amico, Architectural layer recovery for software system understanding and evolution, *Software: Practice and Experience*, vol. 40, pp. 897-916, 2010.
- [S70] M. Shahin, P. Liang, M. Reza, Improving understandability of architecture design through visualization of architectural design decision, *International Conference on Software Engineering Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, 2010.
- [S71] N. Subramanian, L. Chung, Process-oriented metrics for software architecture evolvability, *International Workshop on Principles of Software Evolution*, pp. 65-70, 2003.
- [S72] K. J. Sullivan, W. G. Griswold, Y. Cai, B. Hallen, The structure and value of modularity in software design, *European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 99-108, 2001.
- [S73] M. Svahnberg, An industrial study on building consensus around software architectures and quality attributes, *Information and Software Technology*, vol. 46, pp. 805-818, 2004.
- [S74] L. Tahvildari, K. Kontogiannis, J. Mylopoulos, Quality-driven software re-engineering, *Journal of Systems and Software*, vol. 66, pp. 225-239, 2003.
- [S75] T. Tamai, Y. Torimitsu, Software lifetime and its evolution process over generations, *International Conference on Software Maintenance*, pp. 63-69, 1992.
- [S76] D. Tamzalit, T. Mens, Guiding architectural restructuring through architectural styles, *International Conference and Workshops on Engineering of Computer-Based Systems (ECBS)*, 2010.
- [S77] A. Tang, M. Ali Babar, I. Gorton, J. Han, A survey of architecture design rationale, *Journal of Systems and Software*, vol. 79, pp. 1792-1804, 2006.
- [S78] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, M. Ali-Babar, A Comparative Study of Architecture Knowledge Management Tools, *Journal of Systems and Software*, vol. 83, pp. 352-370, 2009.
- [S79] P. Tarvainen, Adaptability evaluation of software architectures; A case study, *Annual International Computer Software and Applications Conference (COMPSAC)*, pp. 579-584, 2007.
- [S80] M. van den Berg, A. Tang, R. Farenhorst, A constraint-oriented approach to software architecture design, *International Conference on Quality Software*, 2009.
- [S81] W. M. N. Wan-Kadir, P. Loucopoulos, Relating evolving business rules to software design, *Journal of Systems Architecture*, vol. 50, pp. 367-382, 2004.
- [S82] L. Zhu, M. Ali Babar, R. Jeffery, Mining patterns to support software architecture evaluation, *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 25-34, 2004.



## References

- [1] A. Avritzer, E. J. Weyuker, Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews, *Empirical Software Engineering*, vol. 4, (3), pp. 199-215, 1999.
- [2] C. Y. Baldwin, K. B. Clark, *Design rules: Volume 1: The Power of Modularity*, MIT Press Cambridge, ISBN 0-262-02466-7, 2000.
- [3] K. Bennett, Software evolution: past, present and future, *Information and Software Technology*, vol. 38, (11), pp. 673-680, 1996.
- [4] K. Bennett, V. T. Rajlich, Software maintenance and evolution: a roadmap, *Conference on the Future of Software Engineering at International Conference on Software Engineering*, pp. 73-87, 2000.
- [5] I. Borne, S. Demeyer, G. H. Galal, Object-oriented architectural evolution, *European Conference on Object-Oriented Programming 1999*.
- [6] F. P. Brooks, No Silver Bullet, *IEEE Computer*, vol. 20 (4), pp. 10-19, 1987.
- [7] Y. Cai, S. Huynh, An evolution model for software modularity assessment, *International Workshop on Software Quality 2007*.
- [8] L. Chung, *Non-Functional Requirements in Software Engineering*, ISBN 978-0-7923-8666-7, Springer, 2000.
- [9] D. Falessi, R. Capilla, G. Cantone, A value-based approach for documenting design decisions rationale: a replicated experiment, *International Workshop on Sharing and Reusing Architectural Knowledge*, 2008.
- [10] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns: elements of reusable object-oriented software*, ISBN 0-201-63361-2, Addison-Wesley, 1995.
- [11] D. Garlan, *Software architecture: a roadmap*, *Conference on The Future of Software Engineering*, 2000.
- [12] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, *IEEE International Conference on Engineering of Complex Computer Systems (ICECCS) 1998*.
- [13] D. K. Kim, R. France, S. Ghosh, E. Song, A role-based metamodeling approach to specifying design patterns, *IEEE International Computer Software and Applications Conference (COMPSAC)*, 2003.
- [14] B. Kitchenham, Procedures for performing systematic reviews, *Keele University TR/SE-0401/NICTA Technical Report 0400011T*, 2004.
- [15] P. Kruchten, P. Lago, H. van Vliet, Building up and reasoning about architectural knowledge, *International Conference on the Quality of Software Architecture (QoSA) 2006*.
- [16] R. Land, I. Crnkovic, Software Systems In-House Integration: Architecture, Process Practices and Strategy Selection, *Journal of Information and Software Technology*, vol. 49 (5), p419-444, Elsevier, September, 2006.
- [17] N. Lassing, D. Rijsenbrij, H. van Vliet, How well can we predict changes at architecture design time? *Journal of Systems and Software*, vol. 65, (2), pp. 141-153, 2003.
- [18] M. M. Lehman, J. F. Ramil, G. Kahen, Evolution as a noun and evolution as a verb, *Workshop on Software and Organisation Co-Evolution (SOCE)*, pp. 12-13, 2000.
- [19] A. Maccari, Experiences in assessing product family software architecture for evolution, *International Conference on Software Engineering*, pp. 585-592, 2002.
- [20] N. H. Madhavji, J. Fernandez-Ramil, D. Perry, *Software Evolution and Feedback: Theory and Practice*, ISBN 978-0-470-87180-5, John Wiley & Sons, 2006.
- [21] M. Matinlassi, Quality-driven software architecture model transformation, *Working IEEE/IFIP Conference on Software Architecture (WICSA) 2005*.
- [22] M. Mattsson, H. Grahn, F. Mårtensson, Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability, *International Conference on the Quality of Software Architecture (QoSA) 2006*.
- [23] N. Medvidovic, R. N. Taylor, D. S. Rosenblum, An Architecture-Based Approach to Software Evolution, *International Workshop on the Principles of Software Evolution 1998*.
- [24] T. Mens, S. Demeyer, *Software Evolution*, ISBN 978-3-540-76439-7, Springer, 2008.
- [25] C. L. Nehaniv, P. Wernick, Introduction to Software Evolvability, *International Workshop on the Principles of Software Evolution (IWPSE) 2007*.
- [26] R. L. Nord, W. G. Wood, P. Clements, Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method, *Technical Report CMU/SEI-2004-TN-017*, Software Engineering Institute, Carnegie Mellon University, 2004.
- [27] S. T. Redwine Jr, W. E. Riddle, Software technology maturation, *International Conference on Software Engineering*, pp. 189-200, 1985.
- [28] D. Rowe, J. Leaney, Evaluating evolvability of computer based systems architectures-an ontological approach, *International Conference and Workshops on Engineering of Computer-Based Systems (ECBS) pp. 24-28*, 1997.
- [29] D. Rowe, J. Leaney, D. Lowe, Defining systems evolvability-a taxonomy of change, *International Conference and Workshops on Engineering of Computer-Based Systems (ECBS) 1998*.
- [30] K. J. Sullivan, P. Chalasani, V. Sazawal, Software design as an investment activity: a real options perspective, *Real Options and Business Strategy: Applications to Decision Making*, pp. 215-262, 1999.
- [31] S. A. Tonu, A. Ashkan, L. Tahvildari, Evaluating architectural stability using a metric-based approach, *Conference on Software Maintenance and Reengineering (CSMR) 2006*.
- [32] J. van Gurp, J. Bosch, Design erosion: problems and causes, *Journal of Systems and Software*, vol. 61, (2), pp. 105-119, 2002.

- [33] N. H. Weiderman, J. K. Bergey, D. B. Smith, S. R. Tilley, Approaches to Legacy System Evolution, Carnegie-Mellon University, Software Engineering Institute, Technical Report CMU/SEI-97-TR-014, 1997.
- [34] L. Yu, S. Ramaswamy, J. Bush, Symbiosis and Software Evolvability, IT Professional, vol. 10, (4), pp. 56-62, 2008.