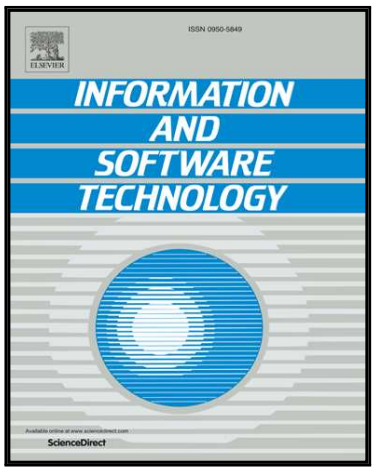

Deriving Architectural Models from Requirements Specifications: a Systematic Mapping Study

Eric Souza, Ana Moreira, Miguel Goulão

PII: S0950-5849(19)30003-5
DOI: https://doi.org/10.1016/j.infsof.2019.01.004
Reference: INFSOF 6090

To appear in: *Information and Software Technology*

Received date: 9 April 2018
Revised date: 12 December 2018
Accepted date: 9 January 2019

Please cite this article as: Eric Souza, Ana Moreira, Miguel Goulão, Deriving Architectural Models from Requirements Specifications: a Systematic Mapping Study, *Information and Software Technology* (2019), doi: https://doi.org/10.1016/j.infsof.2019.01.004

Deriving Architectural Models from Requirements Specifications: a Systematic Mapping Study

Eric Souza[a], Ana Moreira[a], Miguel Goulão[a]

[a]*NOVA LINCS, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa, PORTUGAL*

Abstract

CONTEXT: Software architecture design creates and documents the high-level structure of a software system. Such structure, expressed in architectural models, comprises software elements, relations among them, and properties of these elements and relations. Existing software architecture methods offer ways to derive architectural models from requirements specifications. These models must balance different forces that should be analyzed during this derivation process, such as those imposed by different application domains and quality attributes. Such balance is difficult to achieve, requiring skilled and experienced architects. OBJECTIVE: The purpose of this paper is to provide a comprehensive overview of the existing methods to derive architectural models from requirements specifications and offer a research roadmap to challenge the community to address the identified limitations and open issues that require further investigation. METHOD: To achieve this goal, we performed a systematic mapping study following the good practices from the Evidence-Based Software Engineering field. RESULTS: This study resulted in 39 primary studies selected for analysis and data extraction, from the 2575 initially retrieved. CONCLUSIONS: The major findings indicate that current architectural derivation methods rely heavily on the architects' tacit knowledge (experience and intuition), do not offer sufficient support for inexperienced architects, and lack explicit evaluation mechanisms. These and other findings are synthesized in a research roadmap which results would benefit researchers and practitioners.

Keywords: Software architecture, mapping study, literature review

1. Introduction

Software architecture has been defined in many different ways [1], but at its core it refers to the structure of the software [2] comprising software elements, the externally visible properties of those elements, and the relationships among them [3]. Software architecture supports the analysis of properties of complex systems, and its major tasks are still hard to accomplish, demanding forceful effort, time, and skilled architects. Note that software architecture is different from software design. Software architecture concerns the identification of the architectural elements, their interactions and the constraints satisfying the requirements and serving as a basis for the design, while software design concerns the modularization, detailing the interfaces of the architectural elements, their algorithms and data types needed to support the architectural solution [4]. To create a software architecture model, one must understand the problem, find a solution, and evaluate the final result. *Understand the problem* requires identifying, analyzing and prioritizing the stakeholders' needs, producing the so-called requirements specification. *Find a solution* is related to deriving a software architecture from the requirements specification. This includes decision-making techniques to select, from the existing software architecture alternatives, those architectural elements that best satisfy the stakeholders' goals and the system's qualities. Finally, *evaluate the final result*, or the proposed solution, requires deciding whether, and to what extent, the chosen alternatives solve the problem. Analyzing the software architecture early in the development life cycle can help identifying and mitigating significant technical risks at a minimal cost [5]. Although this may seem a simple activity, creating an architectural model is difficult and requires knowledgeable architects able to find the difficult balance between the different forces imposed by different application domains and quality attributes. In particular, quality attributes may conflict, requiring trade-off analysis of our requirements, and architectural patterns and styles [6]. These trade-offs *pull* the architecture in various directions, leading to a number of architectural choices that would serve the stakeholder needs with varying levels of satisfaction [7]. This issue has led the software engineering community to investigate methodological guidance to move from the problem space (requirements) to the solution space (software architecture design) [8, 9, 10]. Still, few results exist to overcome this gap [9, 10, 11, 12].

This paper focuses on the transition between requirements engineering and software architecture, offering an overview of the current practice for deriving architectural models from requirements specifications. We use Evidence-Based Software Engineering (EBSE) to better understand the problem and the field of research, and to extract and synthesize the results. EBSE provides a rigorous and reliable research methodology,

Email addresses: er.souza@campus.fct.unl.pt (Eric Souza), amm@fct.unl.pt (Ana Moreira), mgoul@fct.unl.pt (Miguel Goulão)

together with auditing tasks to reduce the researcher bias on the results [13, 14]. Two of the core tools for evidence-based studies are *systematic literature reviews*, focusing on identifying the best practices on a given topic based on empirical evidence, and *systematic mapping studies*, aiming at creating a comprehensive overview of a given research area [15, 16].

The goal of this work is **to carry out a systematic mapping study** of the existing methods to derive architectural models from requirements specifications **for the purpose of** classifying them **with respect to** their context, benefits to the user, content, and validation, **in order to** offer a comprehensive overview on methods for software architecture derivation from requirements specifications, and a research roadmap with a list of open issues requiring future research. These constitute the major contributions of this work.

The remainder of this paper is organized as follows. Section 2 describes the research method applied. Section 3 presents the mapping study results from the extracted data and discusses the main observations found. Section 4 highlights and summarizes the major findings of the mapping study. Section 5 discusses the validity threats of this work and elaborates on their mitigation. Section 6 summarizes the major results of the whole study, proposing a research roadmap for the software architecture community. Prior to the planing and execution of our mapping study, we performed a systematic search looking for systematic studies and surveys with similar research questions. Section 7 discusses some of those studies with research questions, despite different, closer to ours. Finally, Section 8 concludes and suggests directions for further work.

2. Research Method

The first step we performed was to search for surveys and secondary studies with similar goals to the one we had in mind: identify existing methods to derive architectural models. This search resulted in several systematic studies which research questions were analyzed to conclude that our goal was not yet addressed (Some of the identified studies are included in the related work section). The following step was then to perform the three phases proposed for a systematic study [14, 16, 17, 18] to offer a comprehensive overview of the existing methods to derive architectural models: *planning*, *conducting*, and *reporting*. The planning phase aims at defining and evaluating the research questions and research protocol the study must comply with. The conducting phase is to carry out the search of the relevant primary studies, and extract and synthesize the data found according to the protocol. Finally, the reporting phase is about the dissemination of the results. The first two phases (planning and conducting) are presented next, while the last phase summarizing the results and revisiting the research questions is discussed in Section 3.

2.1. Planning

To plan a mapping study, we start by formulating the research questions and the search string to run in the digital libraries, next we define the search strategies and the research sources and studies, then we determine how to assess the quality of the studies and specify what data should be extracted from the selected studies, and finally we review the protocol, seeking external reviewers to validate and offer any additional suggestions for improvement.

2.1.1. Formulating the research questions

The definition of the goal of our study follows the **PICOC** [19] structure described in Table 1. This helps us focusing on what matters for the study in hand, guiding also the extraction phase of the process.

Table 1: PICOC Analysis

Population	Papers focusing on the transition from **Requirements** to **Software Architecture**, considering all types of industries, systems and application domains.
Intervention	**Software architecture derivation** methods that use any methodology, tool, technology, and procedures.
Comparison	Not applicable: our intention is to classify the existing derivation methods based on specific criteria nor to compare the methods with other methods or processes.
Outcome	Overview on the context, benefits to the users, content, and validation of the software architecture derivation methods.
Context	Research papers. We are working in a research context with experts in the domain as well as other practitioners, academics, consultants and students.

This lead to the definition of the following research question:

RQ: *What methods have been proposed for the derivation of software architecture models from requirements specifications?*

2.1.2. Formulating the search string

The key terms in our research question led to the definition of the search query. Table 2 shows those terms (first line), as well as the expressions (following lines) that, connected with `AND` operators, form the final search query.

Table 2: Research Query Building

Research Question	What **methods** have been proposed for the **derivation** of **software architecture** models from **requirements** specifications?
Methods	(process OR method OR technique OR approach OR methodology OR framework)
Derivation	(derivation OR decomposition OR "decision-making")
Software Architecture	("software architecture" OR "architectural constraint" OR "architectural pattern" OR "architectural style" OR "architectural view" OR "architectural viewpoint" OR "architectural model" OR "architecture description language" OR "architecture analysis" OR "architectural analysis" OR "architecture documentation")
Requirements	(requirement OR "non-functional" OR "quality attribute")

2.1.3. Defining the search strategies

Our search strategy used two complementary search methods [20]: automatic and manual. The automatic search uses the search terms from Section 2.1.2 to find primary studies on electronic data sources (see Section 2.1.4). With the manual search we followed a forward snowball approach[1] to identify additional primary studies focusing on software architecture.

[1] There are two types of snowball approaches [21]: backward (from the reference lists) and forward (finding citations to the papers found).

2.1.4. Selecting the research sources

The search string was run in four digital libraries, without date restrictions: IEEEXplore, ACM Digital Library, Science Direct, and SpringerLink. These are the libraries covering the most important forums in Computer Science that publish software architecture works, without date restrictions (e.g., journal papers, conference proceedings and workshop papers), including the best conferences and workshop in the research area, such as WICSA, ECSA, and QoSA.

2.1.5. Selecting studies

Inclusion and exclusion criteria were defined to help selecting the relevant studies for analysis and data extraction. We included peer-reviewed papers from journals, conferences and workshops that present methods to design software architecture (I1), relevant studies cited by authors of the papers we read during the conduction process obtained by forward snowball search (I2), and relevant studies suggested by experts on the topic of research (I3) as recommended in [17]. On the other hand, we excluded informal literature (slide shows, conference reviews, informal reports), secondary and tertiary studies (reviews, surveys)[2] and studies from conferences, workshops and journals without peer-review (E1), duplicated studies or studies with the same content (E2), studies that did not answer the research question (E3), studies not written in English (E4), studies not available for download from the source bases and whose authors did not reply to our request (E5), and studies not meeting some quality criteria (E6) (Regarding quality criteria, more details are given in the following section). In cases of studies complementing their authors' previous work, only the most cited[3] were selected, excluding the other studies as duplicate (E2). The application of these exclusion criteria happened in four rounds. In the first round, we analyzed all the candidate studies through title and abstract reading. Next, with the remaining candidate studies, we performed the second round through full text reading, excluding some more papers and included some studies through I2 criterion. In the third round, we analyzed the candidate studies included by I2 criterion through full-text reading. Finally, in the fourth round, we analyzed the candidate studies included by the I3 criterion through full-text reading.

2.1.6. Assessing the quality of the studies

Although there is no agreed definition of what a high level quality study is, there is a common agreement that the quality of the chosen primary studies is critical for obtaining trustworthy results in empirical studies [17]. We defined four quality assessment criteria (QA1–QA4) to be considered when applying the excluding criteria E6, using an approach similar to that in [22] and based on bibliometric impact information. While QA1 uses a set of general and specific criteria (Table 3), QA2 uses the ranking of the publications fora, QA3 uses the papers' citations and QA4 relaxes QA3. Each of these criteria is discussed next.

QA1 is calculated using the *QualityScore* given by *eq (1)*, where the General (G) and Specific (S) assessment factors are summarized in Table 3. The result is a numerical quantification to rank the selected studies.

$$QualityScore = \left\lceil \frac{\sum_{G=1}^{4}}{4} + \left(\frac{\sum_{S=1}^{4}}{4} \times 3 \right) \right\rceil \quad (1)$$

The quality assessment checklist, with G and S composed of four items each and each one with a maximum score of 1, shows a weighted average, where S weights 3 times more than G, as the specific contribution of a study is more important than the general factors. Papers with an overall score >=3 were considered "high" quality studies; papers with a score >=1.5 and <3 were considered acceptable ("medium" quality); and papers with a score <1.5 were considered of lower quality and therefore excluded from further analysis. It is important to clarify that we do not evaluate the quality of the paper itself with this criterion, but only its contributions' alignment with our research questions (derivation of an architectural model from requirements).

Table 3: Quality Assessment Checklist

General Items (G) = 25%	**Specific Items (S) = 75%**
G1: Problem definition and motivation of the study	S1: Method definition
- Explicit Definition (1) - General Definition (0,5) - No Definition (0)	- Formal definition (1) - Semi-formal definition (0,5) - Lack of formalism (0)
G2: Research methodology and organization	S2: Architecture description
- An empirical Methodology (1) - A generalized analysis (0,5) - Lacks any proper methods (0)	- Specifies more than one architectural viewpoint (1) - Describes 1 architectural viewpoint or 1 Architectural Description Language (0,5) - Does not specify any architectural viewpoint nor Architectural Description Language (0)
G3: Contributions of the study refer to the study results	S3: Evaluation of the study
- Explicit w.r.t state-of-the-art (1) - Implicit w.r.t state-of-the-art (0,5) - Not determined w.r.t state-of-the-art (0)	- Formalized empirical evaluation (1) - Some informal evidences are provided (0,5) - Non-justified or ad-hoc validation (0)
G4: Insights and lessons learned from study	S4: Limitations and future implications of the study
- Explicit and technical description (1) - General recommendations (0,5) - Not described (0)	- Formalized empirical evaluations (1) - Some informal evidences are provided (0,5) - Non-justified or ad-hoc validation (0)

QA2 rates papers according to the forums where they were published. It considers "high" for papers published in forums rated A, and "medium" for papers published in forums rated B, according CORE-ERA[4] for conferences, and SJR[5] for journals.

QA3 rates papers according to their citations, giving "high" to papers with more than five citations and "medium" to papers with one to five citations. The citation numbers are collected from Google Scholar.

[2]Excluding these type of studies is common practice [17].

[3]The citation numbers were collected from Google Scholar.

[4]http://portal.core.edu.au/conf-ranks/

[5]https://www.scimagojr.com/journalrank.php

QA4 relaxes QA3 by considering papers published after 2010, which may have fewer citations for being relatively recent[6]. In this case, papers that have potentially high relevance have at least one citation, and papers that have not been cited have potentially medium relevance. To be included in our review, a paper must obtain QA1>=1.5 and its bibliometric impact criteria QA2-QA4 must be "medium" or higher.

2.1.7. Collecting the Data

To promote the understandability of the area and facilitate the data extraction, the research question was decomposed according to the four dimensions recommended by the NIMSAD framework[7] [24]: Context, Benefits to the User, Content, and Validation. Each dimension originated several (sub)research questions, as shown in Table 4.

Table 4: Research question decomposition

Research questions per NIMSAD dimension	**Reasoning**
Context	
What is the main goal of the method?	Even if the main goal of a method is not to address the derivation of architectural models from requirements specifications, we still register it if the issue is discussed.
What are the method's application domains?	The scope of the method must be well-defined, and the application domain helps defining it.
What is the method's starting point?	There are many types of requirements specifications and models. We wish to know what are the requirements specification and models used as input.
Benefits to the users	
What are the method's benefits to the users?	A method improves some aspects of software architecture or facilitates some tasks performed by a software architect. We wish to collect the method's benefits.
What are the limitations of the methods?	All methods have benefits and limitations. We wish to collect the method's limitations as described by their authors.
Content	
What is the coverage of the method with respect to (1) understanding the problem, (2) finding a solution for the problem, and (3) evaluating the solution?	The problem of creating a software architecture model is not different from solving any other problem. We wish to know if the method includes activities for these three phases.
What are the architectural viewpoints proposed by the method?	The complexity of a software architecture requires several views of the solution and different perspectives for different stakeholders. We wish to know what are the viewpoints suggested.
Does the method define a language or notation to represent the produced artifacts?	Each viewpoint is described using some notation or language. We wish to know what are these notations and languages.
What is the level of automation of the supporting tools?	Automation is a way for decreasing the effort of executing the method's activities.
Validation	
How is the method evaluated?	The maturity of the method is directly related to how it was evaluated (e.g., experimentation, case study).
How is the quality of the method's output validated?	Another indicator of the maturity of a method is the evaluation applied the the produced artifacts.

[6] The data ranges used in Q2-Q4 are borrowed from [23], which, although not demanding, are a more inclusive.

[7] NIMSAD (Normative Information Model-based Systems Analysis and Design) is a framework to evaluate methods.

2.1.8. Reviewing the protocol

Once completed, the research protocol must be reviewed by external reviewers to the study. Our protocol was evaluated by three reviewers. This evaluation was based on a set of slides explaining the study, an evaluation form (with questions for each part of the protocol), and the candidate protocol document. The three reviewers inserted their feedback in the form, and we analyzed it. The evaluation form contained the questions in Table 5. The maximum score for each question was 5. In general, the feedback received was very good, as the quality assessment results ranged from 75% (average 4) to 100% (average 5).

The reviewing process led to the refinement of the protocol, particularly: we removed the publication date restriction to be more inclusive with the journals and conferences chosen, and added the quality score assessment to mitigate any potential subjectivity in the evaluation of the general and specific factors. Additionally to the protocol evaluation, (i) we performed a pilot to identify any possible issues in the application of the protocol, and (ii) we used Fabbri's quality checklist [25] as a supplementary quality step. This checklist has a number of questions to evaluate the search string, research protocol, initial selection of studies, final selection of studies, and data extraction. Although no problems in the protocol were found during the execution of the pilot and performing of the checklist, an insight was the usefulness of a bibliographic tool to facilitate the management of all the studies. Thus, the selected studies were cataloged into a bibliography management tool[8], and the data extracted was kept in a spreadsheet workbook.

Table 5: Evaluation Score

#	**Question**	**Average**
1.	Do the research questions cover the specific work objective?	4.6
2.	Are the questions clear?	4.6
3.	Are the digital libraries representative of the area of study?	5
4.	Are you aware of any relevant conference or journal related to this area of study?	*NA**
5.	Are the keywords in the query sufficient to achieve the mapping study objectives?	4.3
6.	Is the query complete (e.g., missing synonyms)?	5
7.	Are the Boolean operators adequate and are they used adequately in the search query?	5
8.	Is the data extraction procedure complete enough to achieve the mapping study objectives?	4.6
9.	Are the inclusion/exclusion criteria complete enough to achieve the mapping study objectives?	5
10.	Are the quality assessment criteria complete enough to achieve the mapping study objectives?	4

* *This is an open question where the reviewers were asked to offer a list of publication fora.*

2.2. Conduction

The execution of the search string (Section 2.1.2) in the four digital libraries retrieved a total of 2575 candidate primary studies, which were collected and imported into the bibliographic tool. The following step was to select the primary studies by applying the inclusion and exclusion criteria, a step that decreased the number of papers to 78 relevant studies. Table 6 summarizes this process. The selected primary studies were read fully, while still applying the exclusion criteria, and the relevant data

[8] Papers tool: `http://www.papersapp.com`

was extracted and added to a spreadsheet workbook previously structured as a form. Next, we applied the quality assessment approach[9], reducing to 39 the total number of studies for final analysis. A synthesis of the data extracted from these 39 papers is described in the next section and the list of all these papers can be found in Appendix A.

Table 6: Application of the filtering criteria

Criteria	IEEE	ACM	Science Direct	Springer	Snowball	Experts
I1	+188	+905	+203	+1259	+0	+0
I2	+0	+0	+0	+0	+14	+0
I3	+0	+0	+0	+0	+0	+6
E1	-11	-22	-47	-91	-0	-0
E2	-9	-104	-10	-145	-4	-1
E3	-149	-775	-134	-999	-3	-3
E4	-0	-0	-0	-4	-0	-0
E5	-0	-0	-1	-1	-0	-0
E6	-12	-1	-3	-6	-2	-0
Total	**7**	**3**	**9**	**13**	**5**	**2**

I1 - Included peer-reviewed papers.
I2 - Included relevant studies cited by authors.
I3 - Included relevant studies suggested by experts.
E1 - Excluded informal literature and secondary and tertiary studies.
E2 - Excluded duplicated studies or studies with the same content.
E3 - Excluded studies that did not answer the RQs.
E4 - Excluded studies that were not written in English.
E5 - Excluded studies that were not available for download.
E6 - Excluded studies according to the quality assessment.

3. Study results

This section starts with a summary of the demographic data for the primary studies and proceeds to discussing the results according to the four NIMSAD dimensions.

3.1. Demographic data

Among the various types of publications, 12 are conference papers ([S1, S2, S4-S8, S18, S24, S25, S29, S30]) and 12 are journal papers ([S9-S17, S32, S35, S38]) both corresponding to a total of 61.4% of the selected primary studies. From the total number of primary studies, 7 are book chapters ([S19, S22, S23, S31, S36, S37, S39], accounting for 17.9% of the studies), 6 are workshop papers ([S3, S20, S21, S27, S33, S34], corresponding to 15.3%) and, finally, 2 are technical reports ([S26, S28], or 5.1% of the total number of selected studies).

3.2. Context

What is the main goal of the method?. From the 39 selected studies, 35 (89.74%) address the derivation of software architecture [S1-S4, S6-S8, S10-S29, S31, S33-S39], 3 (7.7%) concern the understanding of how an architectural model is created in practice [S9, S30, S32], and one (2.56%) [S5] targets design and analysis of quality attributes [S5]. Although most of the studies discuss software architecture derivation, their focus differs significantly, as described in Table 7.

It is important to notice that, although there are some supporting tools for the derivation process (e.g., QuaDAI [S18] and Monarch [S32]), all the decisions reported in the selected primary studies were based on the architects' tacit knowledge (meaning that they are currently based on the experience and intuition of the authors), and the quality of the artifacts were evaluated subjectively, when evaluated at all (this will be further discussed in in Section 3.5, under the question *How is the method evaluated?*).

Table 7: Context of the selected studies.

Selected Study	Context
S1	Relationship between agile methods and architectural modeling.
S2, S6, S12, S18, S28	Use software product line concepts to derive a software architecture.
S2, S3, S4, S11, S14, S15, S18, S38	Use model-driven techniques to derive an architectural model.
S5, S7, S14, S15, S19, S18, S26, S33, S36	Focus on the satisfaction of non-functional requirements at architectural level.
S8, S27, S31	Describe different views to represent the complex software architectures.
S10, S33	Create software architecture from a problem frame specification.
S11, S17, S19, S20, S21, S37	Derive a software architecture from organizational goals specifications.
S13	Introduce an approach for mining and grouping functionalities (architectural modules) from textual descriptions of requirements using text mining techniques
S16, S22, S23, S24, S25	Use aspect-oriented techniques to derive an architectural model.
S29	Show a process for the design, representation, and evaluation of reference architectures.
S34, S35	Facilitate the software architecture design.
S39	Help architects during architectural design

What are the method's application domains?. From the total number of studies, only one focuses on a specific domain (cyber-physical systems) [S38]. The remaining 38 studies (97.4%) do not discuss the methods' fitness (or unfitness) to particular application domains. Instead, the authors limit their approaches to a given paradigm or technological platform (e.g., model-driven development [S2, S3, S4, S11, S14, S15, S18, S38], aspect-oriented software development [S16, S22-S25], software product lines [S2, S6, S12, S18, S28]). Regarding examples or case studies used for evaluation, 20 studies were illustrated with case studies [S2, S4, S6, S7, S10, S11, S13-S17, S20, S22, S24, S25, S29, S32, S35, S38, S39], 10 use examples [S1, S3, S5, S8, S19, S21, S23, S33, S36, S37], 3 report experiments for specific application domains [S12, S18, S36], and 7 were not illustrated [S9, S26-S28, S30, S31, S34].

What is the method's starting point?. The goal is to identify the type of requirements and requirements models used as a starting point. Table 8 summarizes the results, that we discuss next. All the 39 studies address functional requirements, as expected, and from those, 30 (76.9%) address non-functional requirements [S4, S5, S7-S12, S14-S30, S33-S37]. To aggregate these findings we use Sommerville's [26] classification of NFRs

[9] 19 studies were excluded by QA1, four by QA2, and one by QA3.

into Product NFRs (specify the desired characteristics a product must have), Organizational NFRs (derived from organizational policies and procedures), and External NFRs (derived from factors external to the system and its development process). From those 30 studies, 28 (93.3%) consider product NFRs [S4, S5, S7-S12, S14-S16, S18, S19, S21-S30, S33-S37], 9 (30%) consider organizational NFRs [S7, S11, S12, S17, S20, S26, S28, S30, S37], and one (3.3%) considers external NFRs [S7]. Some of the studies consider more than one type of NFR, resulting in a sum of the percentages by type of NFRS greater than 100%. In particular, 7 studies (23.3%) analyze product and organizational NFRs [S7, S11, S12, S26, S28, S30, S37], and one study (3.3%) considers product, organizational and external NFRs [S7]. Figure 1 (left) depicts the distribution of the studies that consider NFRs as input, also showing the overlaps among types of NFRs.

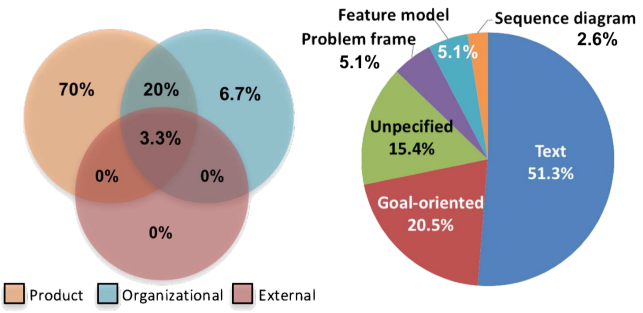

Figure 1: Distribution of NFRs (left) and requirements models (right).

Another relevant aspect is the requirements model used as input, summarized in Table 8. A total of 20 studies (51.3%) use to textual requirements, 8 (20.5%) use goal-oriented models, 6 (15.4%) do not specify any type of model used, 2 use feature models, 2 use problem frames, and 1 uses a sequence diagram model.

Textual Requirements. The high number of studies using textual specifications (in 20 out of 39 studies) may be because industry still uses mostly natural language to specify requirements [S22]. From those, 9 (45%) structure their requirements using use cases [S1-S3, S28, S31, S34, S36, S37, S38], 4 (20%) use intermediary models — e.g., [S14] uses Quality Attribute taxonomy, [S22] uses Requirements Description Language (RDL) [27], [S23] uses Theme/Doc [28], and [S35] uses Component-Bus-System-Property model —, and 2 (10%) create clusters to facilitate modularization and identification of architectural components [S7, S13] ([S13] uses natural language processing algorithms, and [S7] is strongly based on the software architects' experience and intuition). Finally, 5 (25%) of the studies do not give enough information about the structure of the requirements specification [S25, S26, S27, S30, S39].

Goal-oriented models. From the 8 studies using goal-oriented models, 3 use generic goal-oriented models [S4, S12, S19], 3 use i-Star [S11, S17, S20], one uses KAOS [S21], and another one uses v-graph [S24].

Unspecified models. The group of 6 studies ([S5, S8, S9, S16, S29, S32]) that does not specify any type of model used as input for their methods share the idea that architects should analyze and define the critical architectural requirements based on their own experience and, starting with the results of that analysis, begin the architectural modeling process.

3.3. Benefits to the users

What are the benefits the method brings to the users?. From the 39 studies, 30 (76.9%) do not explicitly describe who the users are (target audience) and what are the benefits for them [S1, S2, S4-S7, S9, S11, S12, S14-S30, S32, S35-S37]. In such cases, we consider that the main benefit to the users is the proposal itself. The main benefits of the analyzed studies were the derivation of the software architecture from the requirements (33 out of 39) [S2-S4, S6-S9, S11, S14-S25, S27-S39], followed by understand the requirements specification (12 out of 39) [S3, S7, S9, S13, S22, S26, S28, S30, S31, S34, S35, S37], decision making support (12 out of 39) [S7, S8, S14, S17, S18, S19, S28, S33, S34, S35, S36, S39], architectural Knowledge reuse (6 out of 39) [S1, S8, S14, S18, S21, S25], improve the modularization of software architecture (2 out of 39) [S15, S16], and traceability between requirements and architecture (1 study) [S12]. The result is very interesting because, on the one hand, sharing knowledge between architects is difficult [S8], and, on the other hand, the architectural decision-making process is an important aspect during architectural design. Both have a significant effect on the structure and functionality of the system being developed, hence on the success or failure of the overall software project [29]. However, while systematic processes exist for software architecture model derivation, all architectural decisions have been exclusively based on the architects' tacit knowledge. Thus, further investigation on how to collect and later reuse the architect's tacit knowledge during the decision making process would be useful.

What are the limitations of the method?. From the total number of studies, 32 (82%) did not clearly specify limitations or weaknesses [S1, S3-S7, S9-S20, S23, S25-S30, S32-S35, S37]. These are essential elements for building a sound body of knowledge in a research area. The reason for this lack of information may be due to the level of complexity involved, particularly in finding the balance between the many different forces playing a major role when creating a software architecture [30]. The limitations found are related to the immaturity of the approaches and lack of supporting tool [S21], poor requirements understanding processes [S36], undetailed decision making processes [S8, S31], inconsistency between the artifacts [S2, S22], semantic loss [S24], and lack of traceability between models [S2, S22].

3.4. Content

What is the coverage of the method with respect to (1) understanding the problem, (2) finding a solution for the problem, and (3) evaluating the solution?. In general, a software architecture design approach is composed of an architectural analysis phase to understand the problem, an architectural synthesis

Table 8: Requirements Types and Models Used as Input, where ✓ means requirements type/model used and X means not used.

#	Functional	Product NFRs	Organizational NFRs	External NFRs	Textual	Goal-oriented	Unspecified	Problem frames	Feature model	Sequence model
S1	✓	X	X	X	✓	X	X	X	X	X
S2	✓	X	X	X	✓	X	X	X	X	X
S3	✓	X	X	X	✓	X	X	X	X	X
S4	✓	✓	X	X	X	✓	X	X	X	X
S5	✓	✓	X	X	X	X	✓	X	X	X
S6	✓	X	X	X	X	X	X	X	✓	X
S7	✓	✓	✓	✓	✓	X	X	X	X	X
S8	✓	✓	X	X	X	X	✓	X	X	X
S9	✓	✓	X	X	X	X	✓	X	X	X
S10	✓	✓	X	X	X	X	X	✓	X	X
S11	✓	✓	✓	X	X	✓	X	X	X	X
S12	✓	✓	✓	X	X	✓	X	X	X	X
S13	✓	X	X	X	✓	X	X	X	X	X
S14	✓	✓	X	X	✓	X	X	X	X	X
S15	✓	✓	X	X	X	X	X	X	X	✓
S16	✓	✓	X	X	X	X	✓	X	X	X
S17	✓	X	✓	X	X	✓	X	X	X	X
S18	✓	✓	X	X	X	X	X	X	✓	X
S19	✓	✓	X	X	X	✓	X	X	X	X
S20	✓	X	✓	X	X	✓	X	X	X	X
S21	✓	✓	X	X	X	✓	X	X	X	X
S22	✓	✓	X	X	✓	X	X	X	X	X
S23	✓	✓	X	X	✓	X	X	X	X	X
S24	✓	✓	X	X	X	✓	X	X	X	X
S25	✓	✓	X	X	✓	X	X	X	X	X
S26	✓	✓	✓	X	✓	X	X	X	X	X
S27	✓	✓	X	X	✓	X	X	X	X	X
S28	✓	✓	✓	X	✓	X	X	X	X	X
S29	✓	✓	X	X	X	X	✓	X	X	X
S30	✓	✓	✓	X	✓	X	X	X	X	X
S31	✓	X	X	X	✓	X	X	X	X	X
S32	✓	X	X	X	X	X	✓	X	X	X
S33	✓	✓	X	X	X	X	X	✓	X	X
S34	✓	✓	X	X	✓	X	X	X	X	X
S35	✓	✓	X	X	✓	X	X	X	X	X
S36	✓	✓	X	X	✓	X	X	X	X	X
S37	✓	✓	✓	X	✓	X	X	X	X	X
S38	✓	X	X	X	✓	X	X	X	X	X
S39	✓	X	X	X	✓	X	X	X	X	X

phase to propose solutions to solve the problem, and an architectural evaluation phase to evaluate if the proposed solution solves the problem. Hence, we emphasize that the problem of creating a software architecture model is not different from solving any other problem, where, iteratively, we understand the problem, find a solution, and evaluate that solution. When analyzing the coverage of the methods with respect to these three phases, only 8 studies (20.5%) tackle the three phases [S3, S8, S14, S29-S31, S37, S39]. Including overlaps, 30 (76.9%) aim at understanding the problem and finding a solution [S1-S4, S6-S8, S10-S17, S19, S22, S23, S26, S29-S39], 3 (7.6%) focus on finding a solution (without detailing how to understand the problem) and evaluating it [S9, S18, S28], and 6 (15.3%) focus on creating a solution [S5, S20, S21, S24, S25, S27]. Table 9 shows the coverage of each selected study. Besides the identified gaps, the software architecture methods provide only a coarse-grained description of the proposed method, hence difficult to replicate. This may be because they are currently based on experience and intuition of the authors.

What architectural views does the method use?. Although the various studies use models to describe one or more architectural views, 19 (out of 39) studies do not explicitly name the

Table 9: Coverage of the methods, where ✓ means phase covered and X means phase not covered.

Study	Understanding the problem	Finding a solution for the problem	Evaluating the solution	Study	Understanding the problem	Finding a solution for the problem	Evaluating the solution
S1	✓	✓	X	S21	X	✓	X
S2	✓	✓	X	S22	✓	✓	X
S3	✓	✓	✓	S23	✓	✓	X
S4	✓	✓	X	S24	X	✓	X
S5	X	✓	X	S25	X	✓	X
S6	✓	✓	X	S26	✓	✓	X
S7	✓	✓	X	S27	X	✓	X
S8	✓	✓	✓	S28	X	✓	✓
S9	X	✓	✓	S29	✓	✓	✓
S10	✓	✓	X	S30	✓	✓	✓
S11	✓	✓	X	S31	✓	✓	✓
S12	✓	✓	X	S32	✓	✓	X
S13	✓	✓	X	S33	✓	✓	X
S14	✓	✓	✓	S34	✓	✓	✓
S15	✓	✓	X	S35	✓	✓	X
S16	✓	✓	X	S36	✓	✓	X
S17	✓	✓	X	S37	✓	✓	✓
S18	X	✓	✓	S38	✓	✓	X
S19	✓	✓	X	S39	✓	✓	✓
S20	X	✓	X				

views used [S1, S3-S6, S9-S13, S17, S20, S22, S24, S32-S34, S38, S39]. The remaining 20 studies describe a total of 28 different views: variability view [S2, S18], scenario view [S8, S28, S31], logical view [S8, S19, S28, S31, S35, S36], development view [S8, S14, S28, S31], physical view [S8, S28, S31], process view [S8, S28, S31], structural view [S14, S15, S21, S23], behavioral view [S14, S15, S21, S23], deployment view [S14, S29], internal view [S16], external view [S16], micro view [S17], macro view [S17], functional view [S18, S27], quality view [S18], transformation [S18], modular view [S25, S26, S27, S30], component-connector [S2, S25, S26], allocation view [S26], customer view [S27], application view [S27], conceptual view [S27, S29, S30], realization view [S27], crosscutting view [S29], runtime view [S29], execution view [S30], requirements [S37], and architectural [S37].

A closer analysis of these views suggests that the same view names were used fordifferent purposes. For example, the *development view* was used by [S8] to describe the software modules and their communications and by [S14] to describe the mapping of the software to the hardware. The contrary has also been found, that is, different view names for the same purpose (e.g., [S2] describes the *component-connector view* with the same purpose of the *development view* in [S8]). This and other similar cases seem to indicate lack of a common understanding. To make the analysis possible, we chose the Kruchten's 4+1 Views approach [31] as a baseline method to classify the various views found, "normalizing" those views that shared either the meaning or the name. Although we could have chosen any of the existing proposals, we opted for the widely known 4+1 Views approach that details five different, but complementary, views to represent a software architecture. Table 10 shows the mappings and a rational for each decision, and Figure 2 shows the result of this analysis, for a total of 80 instances of views found and mapped to Krutchen's 4+1 views.

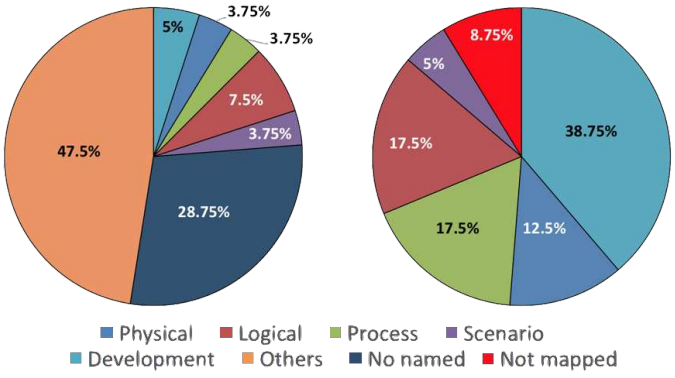

Figure 2: Distributions of architectural views as reported in the primary studies (left) and mapped to the 4+1 views (right).

The visual notation described in [31] was also used for this mapping where, for example, the development view is represented by components (modules and subsystems) and connectors (the communications) among components.

Figure 2-left shows the percentage of instances of the views used by the authors. The total number of instances, for the 28

Table 10: Mapping primary study views to Kruchten 4+1 views; when authors tackle more than one view, we address each sequentially.

Study	Views in primary studies	4+1 Views	Observations
S1	<not named>	Development view	The study describes the organization of software modules using architectural styles, so the mapping to Development view.
S2	Variability view and component-connector view	Not mapped and development view	The variability view shows the common features among different software products. The 4 + 1 approach does not contain this view. The component-connector view describes the organization of software modules (thus, development view).
S3	<not named>	Process view and logical view	The study discusses information flows (hence mapped to process view) and supports functional requirements through class diagrams (hence, logical view).
S4	<not named>	Process view and development view	The study addresses information flows through sequence diagram (process view) and describes the organization of software modules (development view).
S5	<not named>	Logical view	The study describes a logical view with different abstraction levels. The first one abstraction defines the internal behavior of an object with state chart diagrams and another one supports the functional requirements through class diagrams.
S6	<not named>	Not mapped, process view and logical view	The study has a variability view (so not mapped), represents the information flows through collaboration diagrams (process view) and supports functional requirements with class diagrams (logical view).
S7	Micro and macro views	Not mapped and not mapped	The study describes a macro and a micro architecture views, showing the context of software architecture.
S8	Development view, process view, physical view, logical view, and scenario view	Development view, process view, physical view, logical view, and scenario view	The study uses the 4+1 approach.
S9	<not named>	Not mapped	The study does not describe details of the architecture that is generated by the method.
S10	<not named>	Process view and development view	This study describes two views. The first shows the information flows using activity diagrams (process view), and the second shows the organization of software modules using a component-connector diagram (development view).
S11	<not named>	Development view, process view, physical view, logical view, and scenario view	The study uses the 4+1 approach.
S12	<not named>	Development view	The study describes the organization of software modules through the choice of architectural styles.
S13	<not named>	Development view	The study shows the software modules extracted from textual requirements.
S14	Structural view, behavioral view, deployment view, and development view	Development view, process view, physical view, and not mapped	The study does not give details about its development view.
S15	Structural view and behavioral view	Development view and process view	The structural view describes the organization of software modules using component-connector diagram (development view) and the behavioral views shows the information flows through sequence diagram (process view).
S16	Internal view and external view	Not mapped and development view	The internal view is related to source code (not mapped) and the external view shows the organization of software modules using component-connector diagram (development view).
S17	<not named>	Development view	The study shows software modules organization using a component-connector diagram (development view).
S18	Variability view, functional view, quality view, and transformation view	Not mapped, development view, not mapped, and not mapped	Variability view describes the common features among different software products. Quality view represents the relevant system's quality attributes. Transformation view relates to model-driven techniques supporting model transformations. These three views do not map to any of the 4+1 views.
S19	Logical view	Development view	The view shows the software modules organization with a component-connector diagram (development view).
S20	<not named>	Development view	The study shows software modules organization with a component-connector diagram (development view).
S21	Structural view and behavioral view	Development view and process view	The structural view describes the organization of software modules using component-connector diagram (development view) and the behavioral views shows the information flows through state chart diagram (process view).
S22	<not named>	Development view	The study shows software modules organization using a component-connector diagram (development view).
S23	Structural view and behavioral view	Development view and process view	The structural view describes the organization of software modules using component-connector diagram (development view) and the behavioral views shows the information flows with sequence diagram (process view).
S24	<not named>	Development view	The study shows software modules organization using a component-connector diagram (development view).
S25	Modular view and component-connector view	Logical view and development view	The modular view supports the functional requirements through class diagram (logical view) and the component-connector view shows the organization of software modules (development view).
S26	Modular view, component-connector view, and allocation view	Process view, development view, and physical view	The modular view describes the reasoning about the system information flow (process view), component-connector view shows the organization of software modules (development view), and allocation view shows how software elements will be allocated to hardware elements (physical view).
S27	Customer view, application view, functional view, conceptual view, and realization view	Not mapped, scenario view, not mapped, development view, and physical view	The customer view is used to identify the major stakeholders who influence the system development. Functional view intends to describe the externally perceivable properties of the system. Both views are not in 4+1 approach. The application view describes the scenarios necessary to satisfy the customer's need (scenario view). The functional view shows gives an overview of all relevant features (development view). The realization view is related to the selection of hardware to satisfy the system features (physical view).
S28	Development view, process view, physical view, logical view, and scenario view	Development view, process view, physical view, logical view, and scenario view	The study uses the 4+1 approach.
S29	Crosscutting view, runtime view, deployment view, and conceptual view	Not mapped, process view, physical view, and logical view	The crosscutting view shows general information about the reference architecture, such as terms/concepts and variabilities (not mapped). The runtime view shows the dynamic behavior of systems using intern block diagrams (process view). The deployment view describes the hardware (such as, server machines, database servers, and client machines) using package diagrams (physical view). The conceptual view shows the module decomposition using block definition diagram (logical view).
S30	Conceptual view, modular view and execution view	Physical view, development view, and logical view	The conceptual view gives an overview of the system showing the hardware needed to execute the software modules (physical view). The modular view describes the domain specific modules (development view). The execution view details the modules, showing their internal behavior (logical view).
S31	Development view, process view, physical view, logical view, and scenario view	Development view, process view, physical view, logical view, and scenario view	The study uses the 4+1 approach.
S32	<not named>	Development view	The study describes software modules organization with component-connector diagram (development view).
S33	<not named>	Logical view	The study describes functional requirements using class diagrams (logical view).
S34	Development view, process view, logical view, and scenario view	Development view, process view, logical view, and scenario view	The study uses the 4+1 approach, but does not present the physical view .
S35	<not named>	Development view	The study describes software modules organization with component-connector diagram (development view).
S36	<not named>	Logical view	The study describes the architecture through class diagram (logical view).
S37	Requirements and architectural views	Logical and development views	The study describes an architecture through class diagram (logical view) and component-connector model (development view).
S38	¡no named¿	Development views	The study describes the software architecture using AADL (development view).
S39	¡no named¿	Development views	The study describes the architecture through a component diagram (development view).

views in the 39 studies analyzed, is 80[10]. Therefore, 6 out of 80 views (7.5%) are logical views, 4 are development view (5%), followed by physical, process, and scenario views with 3 views each (3.75% each). Besides, 23 views were not named (28.75%), and 38 (47.5%) were described using other names.

Figure 2-right shows that the percentage of studies addressing the logical and development views increased from 7.5% to 17.5% (from 6 to 14 views) and from 5% to 38.7% (from 4 to 31 views), respectively. Seven (8.75%) of the views in the primary studies were not mapped to the 4+1 views. These are, however, alternative and useful views, such as the variability view [S2, S18], the crosscutting view [S29], the customer view [S27], the quality view [S18], the macro view [S7] or internal views [S16].

Does the method define a language or notation to represent the produced artifacts?. We used Jamshidi's categories ("process algebra", "standards" and "others — non-standard"), to classify the studies with respect to the architectural description language used [32].

It is curious to note that in this analysis, the studies were grouped in three sets of the same size. Hence, one third (13 studies corresponding to 33.3%) use standard Architecture Description Languages - ADLs (e.g., UML, SysML, and AADL) [S2, S10, S14, S15, S18, S19, S29, S33-S36, S38, S39], another third does not explicitly use a standard ADL to model the architecture [S3-S5, S11, S16, S17, S20, S22-S25, S32, S37], and the third, does not indicate the architectural description language used [S1, S6-S9, S12, S13, S21, S26-S28, S30, S31] (None of the selected studies used a process algebra ADL). A possible reason for 26 studies not adopting nor defining an ADL could be because an ADL needs to capture design decisions judged fundamental to satisfy different stakeholder concerns [33] and also because it is difficult to capture all the different concerns with a unique language [34].

What is the level of automation of the supporting tools?. Of the 39 selected studies, 18 (46.1%) are partially automated [S2, S4-S6, S10, S12-S16, S18, S19, S22, S32, S35, S36, S38, S39], 8 (20.5%) are not automated [S1, S20, S21, S23-S25, S33, S37], and 13 (33.3%) do not mention supporting tools [S3, S7-S9, S11, S17, S26, S27, S28, S29, S30, S31, S34]. Given the strong dependency between the resulting architecture and the architect's tacit knowledge, it is not surprising that we could not find a complete, fully-fledged automated approach. An important fact found in studies offering some level of automation is that they all propose rules to transform a specific requirements into a specific architectural model. The generated model is then analyzed manually by the architect, and improved where necessary. This requires a considerable effort from the architect. Additionally, there is a lack of tools supporting the activities related to architectural decision making.

[10] For example, consider two studies where one describes a development view and a physical view, and the other describes a development view; the total number of instances of views is three and the number of (types of) views is two.

3.5. Validation

How is the method evaluated?. The goal is to identify the types of empirical approaches used in the evaluation of the approaches proposed by the 39 selected studies. The result shows that 9 (23%) were illustrated with (in some cases simple) examples [S1, S3, S5, S8, S19, S21, S23, S33, S37], 3 (7.6%) were evaluated in experiments [S12, S18, S36], and that 7 (17.9%) were not evaluated [S9, S26-S28, S30, S31, S34]. Although the authors of the remaining 20 (51.2%) studies claim their approaches are evaluated with case studies [S2, S4, S6, S7, S10, S11, S13-S17, S20, S22, S24, S25, S29, S32, S35, S38, S39], these are, in fact, illustrative examples (or else the details of the case study are not given in the paper). The conclusion is that, in general, there is a lack of strong and rigorous empirical evidence. For example, no study describes what hypotheses are being evaluated (case study design), what data is collected, or how the data analysis is performed to check if the purpose of the case study has been reached.

A final fact indicates that 27 (69.2%) of the evaluation approaches were performed in academia [S1-S3, S5, S8, S10, S13-S15, S17-S25, S29, S32-S39], 8 (20.5%) have their roots in industry [S4, S6, S7, S11, S12, S16, S29, S35], and 2 [S29, S35] have academic and industrial case studies.

How is the quality of the method's output validated?. Despite recent studies showing that ATAM (Architecture Tradeoff Analysis Method) [35] is the most used and mature architecture assessment method [36], only 4 (10.2%) of the selected studies use it, or suggest its use [S8, S9, S19, S33]. In addition, [S15] uses ASAAM[11] (Aspectual Software Architecture Analysis Method [38]), [S34] uses SAAM, and S5 uses FDAF (Formal Design Analysis Framework) [39]. We found that 8 (20.5%) of the studies use different scenario-based approaches in an attempt to evaluate the architectural model early in the development[12] [S6, S11, S12, S14, S18, S28, S29, S37] and 21 (53.8%) of the studies do not inform about the evaluation method that can be used with their approaches [S1-S4, S7, S10, S13, S16, S17, S20-S27, S30-S32, S35, S36, S38, S39].

4. Overview of the results

The major findings discussed in some detail for Context (Section 3.2), Benefit to the users (Section 3.3), Content (Section 3.4) and Validation (Section 3.5) are summarized next.

4.1. Context

None of the approaches analyzed are concerned with supporting the architects' design decisions or increasing the quality of the artifacts. All of the approaches are methodology-specific (e.g., aspect-oriented). Around 51% of the methods use textual specifications as their input, and 65% of these structure the requirements using use cases or some other intermediate model. Finally, there is a lack of approaches to build architectural models considering external non-functional requirements.

[11] This is a method based on SAAM [37], a precursor of ATAM.

[12] Architecture defined only conceptually or with a high-level structure.

4.2. Benefit to the users

Given that several studies address more than one benefit (e.g., [S3, S7, S8, S14-S19, S21, S22, S25, S26, S28, S30, S31, S33-S36]), the sum of the number of studies in the following analysis is larger than the total number of analyzed studies. Overall, 33 studies (84.6%) focus on the derivation of the software architecture from the requirements, 12 (30.7%) are concerned with understanding the requirements specification, another 12 aim at providing decision making support, 6 (15.3%) focus on the reuse of architectural knowledge, 2 (5.1%) aim at facilitating the modularization of software architecture, and one (2.7%) focus on providing traceability support between requirements and architecture. Although decision making support and architectural knowledge reuse are considered benefits of some studies, it is noticeable that all the studies are strongly based on the architect's experience. Only 18% of the studies acknowledge their limitations, which include immaturity of the approach, lack of supporting tool, poor requirements understanding process, undetailed decision-making process, inconsistency between the artifacts, semantic loss, and lack of traceability between models.

4.3. Content

A point worth highlighting is the apparent confusion and lack of standardized terms. We argue that the architecture of a software system is a complex task that cannot be described in one single model. Perhaps this is why ISO [40] does not commit itself to any particular views for software architecture description. This lack of standardized terms in software architecture reduces understandability and makes the communication between the involved stakeholders inefficient and error-prone [41]. Our analysis also shows that 66.6% of the studies did not use a standard ADL and that an effort is needed to develop supporting tools (1/3) of studies do not have or do not mention support tools. Standardization of terms would facilitate the adoption of ADLs. We also observed that the methods can hardly be used by novices or less experienced software architects due to lack of detail (the methods are strongly based on the architects' tacit knowledge and lack of systematization) and tool support, as only 46.1% is partially automated, 20.5% is not automated, and 33.3% do not offer information about tools.

4.4. Validation

Case studies is the most used mean for evaluation (mentioned in 51.2% of the studies analyzed). However, in most cases, these case studies are examples to illustrate the approaches. This state of practice should be improved with the adoption of stronger qualitative and quantitative evaluation methods. Regarding validation, 53.8% of the approaches do not provide an explicit way to evaluate the software architecture against the requirements specification.

5. Validity threats and their mitigation

Internal validity. Due to the large number of definitions for the same concept, there is the risk of **not including relevant studies**. The terms used in this study were reviewed by external reviewers as recommended in [17]. We also performed a pilot study to validate the use of the terms and applied snowballing to add relevant studies cited by the authors of included approaches. We used the test-retest strategy on a random subset of selected studies to assess consistency of the selection process as recommended in [17]. Additionally, we included studies suggested by experts, as also recommended in [17]. Regarding the mapping study, we performed two evaluations: we analyzed if the results found in the pilot execution were consistent with the results of the final implementation, and we asked the three external reviewers to check, for a subset of papers, if there were interpretation problems. Subjectivity could be another threat happening during **planning and execution** due to the length of this study. We were so totally immersed in the work that objectivity could be thought as an issue. To mitigate this, we used three external reviewers to evaluate all the phases of the protocol. Additionally, we used Fabbri's best practices checklist [25] to check our work. A final threat concerns the **quality** of the selected papers for analysis and data extraction, as there is no agreement of how this task should be performed. To mitigate this risk we chose only peer-reviewed papers, used the QualityScore approach to reduce the subjectivity of the analysis, and used quality assessment criteria based on bibliometric impact information (approach widely used in systematic reviews published in the literature).

External validity. We could have **missed venues** (e.g., conferences, journals) with relevant published works or have not analyzed relevant articles due to their unavailability. To avoid the first issue, we did not restrict our searchers to venues where more work related to our research was found. We searched in four major digital libraries for relevant related work, and the external reviewers assessed whether these libraries were sufficient for our study and suggested some additional venues to be considered. These venues were used to crosscheck that they were being indexed by the digital libraries. Regarding the issue of unavailable papers, it has been mitigated by requesting the articles directly to the authors through the ResearchGate website and by email. Although the number of articles not available is small (only 2), we can only analyze the studies that we can find.

Conclusion validity. The main conclusion validity threat is the data collection. Since we do not know how the digital libraries' search engines work, we run the risk of getting different results for each search (even because libraries can index new articles daily). Therefore, we ran the search string and, to eliminate the possibility of changes to the list of articles returned by the digital libraries, stored the returned studies in a bibliography management tool[13] for later analysis and data extraction. To mitigate the issue about the data extraction,we decomposed the research question according to the four dimensions recommended by the NIMSAD framework [24] (e.g., Context, Benefits to the User, Content, and Validation) and structured a spreadsheet workbook as a form in order to receive the data

[13] Papers tool: `http://www.papersapp.com`

necessary to answer the research question, as recommended in [17]. In this way, we know precisely what we want to extract from the articles and how to store the extracted data in an organized way.

6. Research roadmap

The previous discussions of the results and major findings highlight several open issues, suggesting worthwhile topics for future research. In particular:

1. **Specific domains.** The studies analyzed are typically methodology-specific (see Section 4.1). However, paradigms and technological platforms change over time and with technological evolution. We believe that developing approaches for specific domains could be seen as a means to collect systems' resources and capabilities to create new and more complex systems, such as systems of systems.
2. **Approaches addressing a wider range of NFRS, particularly external NFRs.** Most studies about the success factors of a software development project indicate that the key critical factors lie in the external non-functional requirements, such as business values satisfaction [42, 43, 44] (see Section 4.1). For example, the Standish Group CHAOS reports state that most software design defects are caused by value-oriented weaknesses [43]. A related relevant topic is to take into account the NFRs (intrinsic) conflicting nature, and study the influence these conflicts have on the architectural decisions and their consequent impact on the final architectural solution. Particularly, more research is needed to investigate the relationships between external NFRs and the software architecture.
3. **Understand the methods' limitations.** Only 18% of studies acknowledge their limitations (see Section 4.2). This may be due to immaturity of the area or due to the level of complexity involved. The fact is that researchers and practitioners should be encouraged to report the limitations and weaknesses of their approaches as these are essential to build a sound body of knowledge.
4. **Standards or common understanding.** The lack of standardized terms, or at least common agreements, reduces understandability and makes the communication between the involved stakeholders inefficient and error-prone [41] (see Section 4.3). Initiatives such as the Software Engineering Institute catalog of different definitions for the term "software architecture" should be more encouraged [45]. This standardization of terms is fundamental for an established body of knowledge, hence beneficial to all software architecture researchers and practitioners (particularly the less experienced ones).
5. **Make explicit tacit knowledge.** Most of the methods can hardly be used by novices or less experienced architects (see Section 4.3). These methods provide only a coarse-grained description of the proposed method, hence making replication impossible or at least too difficult. This may be because they are currently based on the experience and intuition of the authors. Thus, authors must share the process in which a software architecture is built, by providing the steps that must be executed, together with the guidelines and heuristics required to achieve the goal of the method.
6. **Tool support.** From the analyzed approaches, 1/3 does not offer or mention supporting tools (see Section 4.3). Thus, there is a need to develop tool support to facilitate the architects' activities. For example, tools able to suggest the best suited patterns and styles for a given application domain, or to handle specific NFRs to support decision-making activities and decrease the dependence on experienced architects. Such tools could explore artificial intelligence techniques, for example, to create a knowledge base and investigate recommendation algorithms based on the architects' knowledge.
7. **Evaluation methods.** A total of 51.2% of the approaches indicate being evaluated using case studies. However, these case studies are relatively simple illustrations of the authors' approaches (see Section 4.4). This state of practice should be improved with the adoption of stronger qualitative and quantitative methods to support the evaluation.
8. **Requirements satisfaction.** More than half of the studies (53.8%) do not provide an explicit way to validate the resulting software architecture against the requirements specification (see Section 4.4). The existing methods must evolve to facilitate the evaluation of architectural models in the early stages of the development life cycle, or new ones should be created. This should contribute to identify major technical risks, allowing their mitigation at a minimal cost.

7. Related work

We found 31 systematic reviews and mapping studies on software architecture in literature [39-68, 70], focusing on developments in architectural design, analysis and evaluation, These studies include a wide range of topics, such as metrics definition and methods for architectural evaluation [46, 47, 48, 49, 50, 51, 52, 53], quality attributes analysis [54, 51, 52, 55, 56, 57, 58, 59, 60], design strategies for specific types of development environments (e.g., cyber-foraging [61], model-driven development [55], automotive open software architecture [62], component-based architecture [46, 63], service-based architecture [64, 58, 65, 66, 67], decision support for clinical systems [67, 68], computer game architecture [69]), benefits and best practices of use of reference architectures [64, 70, 71, 72], use of architectural patterns [71, 73], architecture erosion [74], and human aspects in software architecture decision making [63, 75]. However, even though several works discuss requirements at the architectural level, they do not address our research questions.

Tofan *et al.* [56] performed a mapping study on architectural decision making, showing that there is much interest in documenting architectural decisions to reduce the "architectural

vaporization knowledge" during the evolution of the architecture. Architectural vaporization is the loss of tacit architectural knowledge throughout the software life cycle, and some reasons for this are listed in [76]. However, these works do not discuss whether there is an interest in documenting architectural decisions to build new similar systems. We believe that this type of knowledge can facilitate software development and reduce the involved risks. Finally, Li *et al.* study the application of knowledge-based approaches in software architecture, and discuss the lack of supporting tools that use a knowledge base for these activities [77]. Thus, reusing knowledge about decision-making and evaluation is still a topic deserving further research.

8. Conclusions

The goal of this paper is to provide a comprehensive overview of the existing methods for the derivation of software architecture models from requirements specifications. To achieve this, we performed a systematic mapping study to find empirical evidence about software architecture derivation methods, classifying them with respect to their context, benefits to the user, content and validation. The end result is an overview of the current practice on deriving architectural models from a requirements specifications, both in industry and academia. The evidence found indicates that the studied methods have been strongly based on the architects' tacit knowledge, without enough systematization and tool support. Also, the lack of standardized terms in the definitions of software architecture concepts was noticeable. Finally, early architecture evaluations are not sufficiently supported by the existing architecture evaluation methods. These and other open issues identified were used to offer a research agenda.

In the near future work, we will focus our research on the systematization of the manner in which a software architecture is built (corresponding to item 5 of the research roadmap). Also, we will contribute to improve the state of practice with the conduction of controlled experiments to evaluate some methods for software architecture derivation (item 7 from the research roadmap), as recommended by the Evidence-Based Software Engineering community.

Acknowledgments

We thank NOVA LINCS Software Engineering team for validating the protocol and results of the study described here, and Programa Ciência sem Fronteiras (Ref. 99999.009047/2013-01) and NOVA LINCS Research Laboratory (Ref. UID/CEC/04516/2013) for their financial support.

References

[1] L. Hohmann, Beyond Software Architecture: Creating and Sustaining Winning Solutions, 1st edition, 2003.

[2] D. Garlan, Software architecture: a travelogue, in: the, ACM Press, New York, New York, USA, 2014, pp. 29–39.

[3] L. Bass, P. C. Clements, R. Kazman, Software architecture in practice, SEI series in software engineering, Addison-Wesley, Boston, 2nd edition edition, 2003.

[4] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, ACM SIGSOFT Software engineering notes 17 (1992) 40–52.

[5] A. Zalewski, S. Kijas, Beyond ATAM: Architecture Analysis in the Development of Large Scale Software Systems 4758 (2013) 683–697.

[6] A. Rashid, A. Moreira, J. Araújo, Modularisation and composition of aspectual requirements, in: Proceedings of the 2nd International Conference on Aspect-oriented Software Development, AOSD'03, ACM, USA, 2003, pp. 11–20.

[7] A. Moreira, A. Rashid, J. Araujo, Multi-dimensional separation of concerns in requirements engineering, in: 13th IEEE International Conference on Requirements Engineering (RE'05), pp. 285–296.

[8] C. Vetterli, W. Brenner, F. Uebernickel, C. Petrie, From palaces to yurts: Why requirements engineering needs design thinking, IEEE Internet Computing 17 (2013) 91–94.

[9] P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, I. Mistrík, Relating software requirements and architectures, Springer Science & Business Media, 2011.

[10] R. C. de Boer, H. v. Vliet, On the similarity between requirements and architecture, The Journal of Systems and Software 82 (2009) 544–550.

[11] M. Galster, A. Eberlein, M. Moussavi, Transition from requirements to architecture: A review and future perspective, in: Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on, IEEE.

[12] A. Alebrahim, Bridging the Gap between Requirements Engineering and Software Architecture: A Problem-Oriented and Quality-Driven Method, Springer, 2017.

[13] B. A. Kitchenham, T. Dybå, M. Jørgensen, Evidence-Based Software Engineering., ICSE (2004) 273–281.

[14] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain 80 (2007) 571–583.

[15] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: EASE'08: Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering, British Computer Society, 2008.

[16] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Information and Software Technology 64 (2015) 1–18.

[17] B. A. Kitchenham, Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report, Department of Computer, Science University of Durham, Durham, UK, 2007.

[18] C. Wohlin, Writing for synthesis of evidence in empirical software engineering, in: the 8th ACM/IEEE International Symposium, ACM Press, New York, New York, USA, 2014, pp. 1–4.

[19] M. Petticrew, H. Roberts, Systematic Reviews in the Social Sciences: A practical guide, Blackwell Publishing, 2005.

[20] H. Zhang, M. A. Babar, P. Tell, Identifying relevant studies in software engineering, Information and Software Technology 53 (2011) 625–637.

[21] S. Jalali, C. Wohlin, Systematic literature studies: database searches vs. backward snowballing, in: the ACM-IEEE international symposium, ACM Press, New York, New York, USA, 2012, pp. 29–10.

[22] A. Ahmad, P. Jamshidi, C. Pahl, Protocol for Systematic Literature Review, Technical Report, Dublin City University, Dublin - Ireland, 2012.

[23] D. Bombonatti, M. Goulão, A. Moreira, Synergies and tradeoffs in software reuse–a systematic mapping study, Software: practice and experience 47 (2017) 943–957.

[24] N. Jayaratna, Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framework, McGraw-Hill, Inc., New York, NY, USA, 1994.

[25] S. Fabbri, K. Felizardo, F. Ferrari, E. Hernandes, F. Octaviano, E. Nakagawa, J. Maldonado, Externalising tacit knowledge of the systematic review process, IET software 7 (2013) 298–307.

[26] I. Sommerville, G. Kotonya, Requirements Engineering: Processes and Techniques, John Wiley & Sons, Inc., 1998.

[27] R. Chitchyan, A. Rashid, P. Rayson, R. Waters, Semantics-based composition for aspect-oriented requirements engineering, in: Proceedings of the 6th international conference on Aspect-oriented software development, ACM, pp. 36–48.

[28] C. Siobhan, B. Elisa, Aspect-oriented analysis and design: The theme approach, 2005.

[29] P. Petrov, R. L. Nord, U. Buy, Probabilistic Macro-Architectural Decision Framework, in: the 2014 European Conference, ACM Press, New York, New York, USA, 2014, pp. 1–8.
[30] D. Falessi, G. Cantone, P. Kruchten, Do Architecture Design Methods Meet Architects' Needs?, in: 2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07), IEEE, 2007, pp. 5–5.
[31] P. Kruchten, The 4+1 View Model of architecture, IEEE Software 12 (1995) 42–50.
[32] P. Jamshidi, M. Ghafari, A. Ahmad, C. Pahl, A Framework for Classifying and Comparing Architecture-centric Software Evolution Research, in: 2013 17th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, 2013, pp. 305–314.
[33] N. Medvidovic, E. M. Dashofy, R. N. Taylor, Moving architectural description from under the technology lamppost, Information and Software Technology 49 (2007) 12–31.
[34] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, A. Tang, What Industry Needs from Architectural Languages: A Survey, IEEE Transactions on Software Engineering 39 (2013) 869–891.
[35] R. Kazman, M. H. Klein, P. C. Clements, ATAM: Method for architecture evaluation, Technical Report, 2000.
[36] C. Catal, M. Atalay, A Systematic Mapping Study on Architectural Analysis, in: International Conference on Information Technology: New Generations (ITNG), IEEE, 2013, pp. 661–664.
[37] R. Kazman, L. Bass, G. Abowd, M. Webb, SAAM: a method for analyzing the properties of software architectures, in: Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on, IEEE Computer Society Press, 1994, pp. 81–90.
[38] B. Tekinerdogan, ASAAM: Aspectual Software Architecture Analysis Method., WICSA (2004) 5–14.
[39] L. Dai, K. Cooper, Formal design analysis framework: an aspect-oriented architectural framework, University of Texas at Dallas, 2005.
[40] ISO/IEC/IEEE 42010, International electrotechnical commission (iec), institute of electrical and electronics engineers(ieee): Systems and software engineering - architecture description (iso/iec/ieee 42010) (2011).
[41] K. Smolander, What is included in software architecture? a case study in three software organizations, in: IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, IEEE.
[42] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, P. Grünbacher, Value-based software engineering, Springer Science & Business Media, 2006.
[43] Standish Group, Standish Group Website, http://www.standishgroup.com, 2017.
[44] E. Souza, A. Moreira, J. Araújo, S. Abrahão, E. Insfran, D. S. da Silveira, Comparing business value modeling methods: A family of experiments, Information and Software Technology (2018).
[45] Software Engineering Institute, What is your definition of software architecture?, https://goo.gl/PBCLWR, 2010.
[46] M. Abdellatief, A. B. M. Sultan, A. A. A. Ghani, M. A. Jabar, A mapping study to investigate component-based software system metrics, Journal of systems and software 86 (2013) 587–603.
[47] E. Murugesupillai, B. Mohabbati, D. Gasević, A preliminary mapping study of approaches bridging software product lines and service-oriented architectures, in: Proceedings of the 15th International Software Product Line Conference, Volume 2, ACM, p. 11.
[48] F. Nikpay, R. Ahmad, B. D. Rouhani, S. Shamshirband, A systematic review on post-implementation evaluation models of enterprise architecture artefacts, Information Systems Frontiers (2016) 1–20.
[49] L. B. R. de Oliveira, E. Y. Nakagawa, A systematic review on service-oriented reference models and service-oriented reference architectures, Available on https://goo.gl/UaQjfR, 2010.
[50] B. J. Williams, J. C. Carver, Characterizing software architecture changes: A systematic review, Information and Software Technology 52 (2010) 31–51.
[51] S. Stevanetic, U. Zdun, Software metrics for measuring the understandability of architectural structures: a systematic mapping study, in: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, ACM, p. 21.
[52] H. Koziolek, Sustainability evaluation of software architectures: a systematic review, in: Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS, ACM, pp. 3–12.
[53] M. Palacios, J. García-Fanjul, J. Tuya, Testing in service oriented architectures with dynamic binding: A mapping study, Information and Software Technology 53 (2011) 171–189.
[54] G. Me, C. Calero, P. Lago, A long way to quality-driven pattern-based architecting, in: European Conference on Software Architecture, Springer, pp. 39–54.
[55] D. Ameller, X. Burgués, O. Collell, D. Costal, X. Franch, M. P. Papazoglou, Development of service-oriented architectures using model-driven development: A mapping study, Information and Software Technology 62 (2015) 42–66.
[56] D. Tofan, M. Galster, P. Avgeriou, W. Schuitema, Past and future of software architectural decisions–a systematic mapping study, Information and Software Technology 56 (2014) 850–872.
[57] A. Arshad, M. Usman, Security at software architecture level: a systematic mapping study, in: Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on, IET, pp. 164–168.
[58] S. Mahdavi-Hezavehi, M. Galster, P. Avgeriou, Variability in quality attributes of service-based software systems: A systematic literature review, Information and Software Technology 55 (2013) 320–343.
[59] M. M. Ahmed, S. Letchmunan, A systematic literature review on challenges in service oriented software engineering, International Journal of Software Engineering and Its Applications 9 (2015) 173–186.
[60] M. Razavian, P. Lago, A frame of reference for soa migration, in: European Conference on a Service-Based Internet, Springer, pp. 150–162.
[61] G. Lewis, P. Lago, Architectural tactics for cyber-foraging: Results of a systematic literature review, Journal of Systems and Software 107 (2015) 158–186.
[62] S. Dersten, J. Axelsson, J. Froberg, Effect analysis of the introduction of autosar: a systematic literature review, in: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, IEEE, pp. 239–246.
[63] J. Cruz-Benito, R. Therón, F. J. García-Peñalvo, Software architectures supporting human-computer interaction analysis: A literature review, in: International Conference on Learning and Collaboration Technologies, Springer, pp. 125–136.
[64] F. Aulkemeier, M. Schramm, M.-E. Iacob, J. Van Hillegersberg, A service-oriented e-commerce reference architecture, Journal of theoretical and applied electronic commerce research 11 (2016) 26–45.
[65] N. Alshuqayran, N. Ali, R. Evans, A systematic mapping study in microservice architecture, in: Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on, IEEE, pp. 44–51.
[66] Q. Gu, P. Lago, Exploring service-oriented system engineering challenges: a systematic literature review, Service Oriented Computing and Applications 3 (2009) 171–188.
[67] S. R. Loya, K. Kawamoto, C. Chatwin, V. Huser, Service oriented architecture for clinical decision support: a systematic review and future directions, Journal of medical systems 38 (2014) 140.
[68] L. Tabares, J. Hernandez, I. Cabezas, Architectural approaches for implementing clinical decision support systems in cloud: a systematic review, in: Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on, IEEE, pp. 42–47.
[69] M. Zhu, A. I. Wang, H. Guo, From 101 to nnn: a review and a classification of computer game architectures, Multimedia systems 19 (2013) 183–197.
[70] S. Martinez-Fernandez, P. S. M. Dos Santos, C. P. Ayala, X. Franch, G. H. Travassos, Aggregating empirical evidence about the benefits and drawbacks of software reference architectures, in: Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on, IEEE, pp. 1–10.
[71] I. Lytra, S. Sobernig, U. Zdun, Architectural decision making for service-based platform integration: A qualitative multi-method study, in: Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on, IEEE, pp. 111–120.
[72] M. Guessi, E. Y. Nakagawa, F. Oquendo, J. C. Maldonado, Architectural description of embedded systems: a systematic review, in: Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems, ACM, pp. 31–40.
[73] J. Juziuk, D. Weyns, T. Holvoet, Design patterns for multi-agent systems: a systematic literature review, in: Agent-Oriented Software Engineering,

Springer, pp. 79–99.

[74] N. Qureshi, M. Usman, N. Ikram, Evidence in software architecture, a systematic literature review, in: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, ACM, pp. 97–106.

[75] A. Tang, M. Razavian, B. Paech, T.-M. Hesse, Human aspects in software architecture decision making: a literature review, in: Software Architecture (ICSA), 2017 IEEE International Conference on, IEEE, pp. 107–116.

[76] K. C. Joshi, H. Hora, Software Architecture & Architectural Design Decision: Impact of Architectural Vaporization, volume III, International Journal of Advanced Engineering Technology, 2012.

[77] Z. Li, P. Liang, P. Avgeriou, Application of knowledge-based approaches in software architecture: A systematic mapping study, Information and Software Technology 55 (2013) 777–794.

Appendix A: Selected Studies

Selected Studies

[S1] Z. Durdik, Towards a process for architectural modelling in agile software development, in: Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS, ACM, pp. 183–192, 2011.

[S2] A. S. Nascimento, C. M. Rubira, R. Burrows, F. Castor, A model-driven infrastructure for developing product line architectures using cvl, in: Software Components, Architectures and Reuse (SBCARS), 2013 VII Brazilian Symposium on, IEEE, pp. 119–128.

[S3] G. Loniewski, A. Armesto, E. Insfran, An architecture-oriented model-driven requirements engineering approach, in: Model-Driven Requirements Engineering Workshop (MoDRE), 2011, IEEE, pp. 31–38.

[S4] F. Montero, E. Navarro, Atrium: Software architecture driven by requirements, in: Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on, IEEE, pp. 230–239.

[S5] L. Dai, K. Cooper, Modeling and analysis of non-functional requirements as aspects in a uml based architecture design, in: Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on, IEEE, pp. 178–183.

[S6] P. Sochos, M. Riebisch, I. Philippow, The feature-architecture mapping (farm) method for feature-oriented development of software product lines, in: Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on, IEEE.

[S7] P. Petrov, U. Buy, R. L. Nord, The need for a multilevel context-aware software architecture analysis and design method with enterprise and system architecture concerns as first class entities, in: Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on, IEEE, pp. 147–156.

[S8] E. Woods, N. Rozanski, Using architectural perspectives, in: Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on, IEEE, pp. 25–35.

[S9] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, P. America, A general model of software architecture design derived from five industrial approaches, Journal of Systems and Software 80 (2007) 106–126.

[S10] P. Colombo, F. Khendek, L. Lavazza, Bridging the gap between requirements and design: An approach based on problem frames and sysml, Journal of Systems and Software 85 (2012) 717–745.

[S11] J. Castro, M. Lucena, C. Silva, F. Alencar, E. Santos, J. Pimentel, Changing attitudes towards the generation of architectural models, Journal of Systems and Software 85 (2012) 463–479.

[S12] J. Kim, S. Park, V. Sugumaran, Drama: A framework for domain requirements analysis and modeling architectures in software product lines, Journal of Systems and Software 81 (2008) 37–55.

[S13] A. Casamayor, D. Godoy, M. Campo, Functional grouping of natural language requirements for assistance in architectural software design, Knowledge-Based Systems 30 (2012) 78–86.

[S14] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, P. Aho, Knowledge based quality-driven architecture design and evaluation, Information and Software Technology 52 (2010) 577–601.

[S15] P. Sánchez, A. Moreira, L. Fuentes, J. Araújo, J. Magno, Model-driven development for early aspects, Information and Software Technology 52 (2010) 249–273.

[S16] J. Pérez, N. Ali, J. A. Carsı, I. Ramos, B. Álvarez, P. Sanchez, J. A. Pastor, Integrating aspects in software architectures: Prisma applied to robotic tele-operated systems, Information and Software Technology 50 (2008) 969–990.

[S17] J. Castro, M. Kolp, J. Mylopoulos, Towards requirements-driven information systems engineering: the tropos project, Information systems 27 (2002) 365–389.

[S18] J. González-Huerta, E. Insfrán, S. Abrahão, Defining and validating a multimodel approach for product architecture derivation and improvement, in: International Conference on Model Driven Engineering Languages and Systems, Springer, pp. 388–404, 2013.

[S19] L. Chung, S. Supakkul, N. Subramanian, J. L. Garrido, M. Noguera, M. V. Hurtado, M. L. Rodríguez, K. Benghazi, Goal-oriented software architecting, in: Relating Software Requirements and Architectures, Springer, 2011, pp. 91–109.

[S20] M. Lucena, J. Castro, C. Silva, F. Alencar, E. Santos, J. Pimentel, A model transformation approach to derive architectural models from goal-oriented requirements models, in: On the Move to Meaningful Internet Systems: OTM 2009 Workshops, Springer, pp. 370–380.

[S21] S. Tang, X. Peng, Y. Yu, W. Zhao, Goal-directed modeling of self-adaptive software architecture, Enterprise, Business-Process and Information Systems Modeling (2009) 313–325.

[S22] R. Chitchyan, M. Pinto, A. Rashid, L. Fuentes, Compass: composition-centric mapping of aspectual requirements to architecture, Transactions on aspect-oriented software development IV (2007) 3–53.

[S23] P. Sánchez, L. Fuentes, A. Jackson, S. Clarke, Aspects at the right time, in: Transactions on aspect-oriented software development IV, Springer, 2007, pp. 54–113.

[S24] L. Silva, T. Batista, A. Garcia, A. Medeiros, L. Minora, On the symbiosis of aspect-oriented requirements and architectural descriptions, Early Aspects: Current Challenges and Future Directions (2007) 75–93.

[S25] R. Cordero, I. Salavert, J. Torres-Jiménez, Designing aspectual architecture views in aspect-oriented software development, Computational Science and Its Applications-ICCSA 2006 (2006) 726–735.

[S26] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, B. Wood, Attribute-driven design (ADD), version 2.0, Technical Report, Carnegie-Mellon University Pittsburgh PA Software Engineering Institute, 2006.

[S27] P. America, E. Rommes, H. Obbink, Multi-view variation modeling for scenario analysis, in: International Workshop on Software Product-Family Engineering, Springer, pp. 44–65, 2003.

[S28] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, F. Peruzzi, The architecture based design method, Technical Report, Carnegie-Mellon University Pittsbutgh PA Software Engineering Institute, 2000.

[S29] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, F. Oquendo, Consolidating a process for the design, representation, and evaluation of reference architectures, in: Software Architecture (WICSA), 2014 IEEE/IFIP Conference on, IEEE, pp. 143–152.

[S30] D. Soni, R. L. Nord, C. Hofmeister, Software architecture in industrial applications, in: Software Engineering, 1995. ICSE 1995. 17th International Conference on, IEEE, pp. 196–196.

[S31] P. Kruchten, The rational unified process: an introduction, Addison-Wesley Professional, 2004.

[S32] H. Bagheri, K. Sullivan, Model-driven synthesis of formally precise, stylized software architectures, Formal Aspects of Computing 28 (2016) 441.

[S33] A. Alebrahim, S. Fassbender, M. Filipczyk, M. Goedicke, M. Heisel, Towards systematic selection of architectural patterns with respect to quality requirements, in: Proceedings of the 20th European Conference on Pattern Languages of Programs, ACM, pp. 40, 2015.

[S34] L. Tian, L. Zhang, B. Zhou, G. Qian, A gradually proceeded software architecture design process, in: Software Process Workshop, Springer, pp. 192–205, 2005.

[S35] P. Grünbacher, A. Egyed, N. Medvidovic, Reconciling software requirements and architectures with intermediate models, Software & Systems

Modeling 3 (2004) 235–253.

[S36] O. Räihä, H. Kundi, K. Koskimies, E. Mäkinen, Synthesizing architecture from requirements: A genetic approach, in: Relating Software Requirements and Architectures, Springer, 2011, pp. 307–331.

[S37] K. Pohl, E. Sikora, The co-development of system requirements and functional architecture, in: Conceptual Modelling in Information Systems Engineering, Springer, 2007, pp. 229–246.

[S38] R. F. Passarini, J.-M. Farines, J. M. Fernandes, L. B. Becker, Cyber-physical systems design: transition from functional to architectural models, Design Automation for Embedded Systems 19 (2015) 345–366.

[S39] A. Tang, H. Van Vliet, Software architecture design reasoning, in: Software Architecture Knowledge Management, Springer, 2009, pp. 155–174.