A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research

Pooyan Jamshidi [1], Mohammad Ghafari [2], Aakash Ahmad [1], Claus Pahl [1]
[1] Lero - The Irish Software Engineering Research Centre
School of Computing, Dublin City University, Ireland
[2] DeepSE Group @ DEI - Politecnico di Milano, Italy
[1] {pooyan.jamshidi|ahmad.aakash|claus.pahl}@computing.dcu.ie, [2] ghafari@elet.polimi.it

Abstract—***Context*: Software systems are increasingly required to operate in an open world, characterized by continuous changes in the environment and in the prescribed requirements. Architecture-centric software evolution (ACSE) is considered as an approach to support software adaptation at a controllable level of abstraction in order to survive in the uncertain environment. This requires evolution in system structure and behavior that can be modeled, analyzed and evolved in a formal fashion. Existing research and practices comprise a wide spectrum of evolution-centric approaches in terms of formalisms, methods, processes and frameworks to tackle ACSE as well as empirical studies to consolidate existing research. However, there is no unified framework providing systematic insight into classification and comparison of state-of-the-art in ACSE research.**

***Objective*: We present a *taxonomic scheme* for a *classification* and *comparison* of existing ACSE research approaches, leading to a reflection on areas of future research.**

***Method*: We performed a systematic literature review (SLR), resulting in 4138 papers searched and 60 peer-reviewed papers considered for data collection. We *populated* the taxonomic scheme based on a quantitative and qualitative *extraction* of data items from the included studies.**

***Results*: We identified five main classification categories: (i) *type of evolution*, (ii) *type of specification*, (iii) *type of architectural reasoning*, (iv) *runtime issues*, and (v) *tool support*. The selected studies are compared based on their claims and supporting evidences through the scheme.**

***Conclusion*: The classification scheme provides a critical view of different aspects to be considered when addressing specific ACSE problems. Besides, the consolidation of the ACSE evidences reflects current trends and the needs for future research directions.**

Keywords- Architecture-Centric Software Evolution; Evidence-Based and Empirical Study; Systematic Literature Review

I. INTRODUCTION

Modern software systems are increasingly required to operate in an open world [27], characterized by frequent and unpredictable change in the environment in which they are functioning and in the requirements they have to meet. Considering existing research [1, 6, 18] and practice [23, 28], software architectures provide a sound basis to smoothly evolve software and dynamically adapt it to provide expected services. Architecture-centric software evolution [1, 6, 10] allows an appropriate abstraction to model, analyze and execute software evolution in a controllable and manageable fashion.

Traditionally, software architecture is considered as an appropriate abstraction level in the early stages of software development to better understand requirements, systematically communicate with the stakeholders and objectively reason about qualities. Architecture models also help to crystallize design decisions and evaluate the tradeoffs among them [1, 4]. This role of software architecture also contributes to *control* the evolution [21] in order to avoid degradation as *erosion* [5], *drifts* [4] as well as architecture *pendency* [16].

Software architecture models not only facilitate software development, but also extend their lifetime to run-time to support dynamic software adaptation [30] and continuous verification [24]. This promotes dynamic architectures as a means to evolve software systems at runtime [S44][1]. Through *reflective middleware*, software architecture models can be actively connected to the running systems by providing facilities to change the computation logic of a system.

We observed that existing studies for evidence-based consolidation of ACSE research has focused on surveying [6] and characterizing [10] software architecture evolvability. They also focus on comparison of the support for architecture evolution [17] and classification of dynamic aspects of architecture [S32] as well as taxonomic frameworks [18]. These studies provide insight in the potential use of software architecture in software evolution. However, we could not find any evidence to empirically synthesize the collective impact of existing literature.

The objective of this paper is to *systematically (i) identify the focus of current research and (ii) classify the claims made for ACSE and available evidences for these claims, and; (iii) provide a holistic comparison to analyze the potentials and limitations in current approaches that (iv) outline hypotheses for future research.*

To achieve this, we performed a systematic literature review. We adopted and tailored an integrated approach [11, 26] to extract a coded schema by which we can systematically review state-of-the-art of ACSE. Based on this, we collected data from selected studies to answer five questions: (i) what types of *evolutions* are supported? (ii) which *formalisms* are required? (iii) how *architectural reasoning* is applied? (iv) which *execution environments* are needed? (v) what *tool support* is available?.

Having answered the research questions, we highlight areas and hypotheses for future research. We report our observations of the synthesis on the extracted data. Extended materials that were used for the study comprising a *protocol*, search results, quality assessments and *lessons* learned as well as the *extracted data* are available at [25].

[1] Please note that notation [S_N] refers to the primary studies in Table 5.

The rest of the paper is structured as follows: Section II compares and contrasts related work to justify the needs and scope of this review. Section III outlines the methodology we adopted in this SLR. In Section IV we explain study planning. The first contribution of this paper, a classification framework for ACSE approaches, is presented in Section V. This allows us to synthesize the data extracted from the primary studies and interpret this data answering the research questions in Section VI. Section VII identifies future research directions based on the results and it also discusses the limitations of our study. Finally, Section VII presents the conclusions.

II. Related Work

In recent years, evidence-based and empirical research in software engineering gained a considerable momentum [3]. In the context of ACSE, we observed that existing studies have focused on evolvability analysis [6], change characterization [10] and classification of approaches [S32], as summarized in Table 1. These are discussed below in order to justify the needs for this review.

A. Systematic Reviews on Software Architecture Evolution

Breivold [6] systematically reviews software architecture evolvability analysis research. The objective is to obtain an overview of approaches in analyzing and improving software evolvability at architectural level, and investigate impacts on research and practice. This survey presents a synthesis of 82 primary studies. Their focus is on evolvability in general and analyzability, architectural integrity and changeability in particular. Therefore, they synthesize different sets of primary studies; our work focuses on the studies which formally specify software architecture in order to enable a controllable evolution.

Bradbury et al. [S32] present a set of classification criteria for the comparison of dynamic software architectures based on change type, process and infrastructure. They synthesize 14 formal specification approaches to discover similarities and differences. In contrast to our proposal, they dedicatedly compare the dynamic reconfigurations and architectural formalisms to gain a deeper understanding of run-time software architecture evolution, while not focusing on the other dimensions such as design-time evolutionary aspects.

Williams and Carver [10] propose a change characterization scheme and systematically classify different approaches on how to distinguish and characterize software architecture changes and software impact analysis. This scheme works as a decision tree providing support for developers to assess the impact of a proposed change and decide whether it is feasible to implement the change.

We also conducted an SLR [31] to classify studies according to evidences that enable application and acquisition of evolution reuse-knowledge in ACSE. We summarize the contribution of these studies to better position the contribution of this paper in Table 1.

B. Comparative Studies on Description Languages

There are several surveys on architecture description languages (ADLs) [8, 9, 17]. More recently, Medvidovic et al. [12] look at recent developments and other aspects of architecture description which should be treated by researchers and practitioners. Our paper extends some of the findings related to evolution support in ADLs illustrated in their work. Mens et al. [1] also note the importance of software architecture description to accommodate changes when there is a need to evolve critical software systems. They highlight key problem areas in ACSE and review the existing promising proposals to tackle them which helped us in the categorization process in our thematic mappings.

C. Taxonomies on Software Evolution

Although not directly related to the ACSE study presented in this paper, some taxonomies of software change [18, 19] are proposed trying to answer the why, how, what, when and where of software evolution.

TABLE 1. SYSTEMATIC REVIEWS ON ACSE

Study Reference	Study Focus	Change Time	Number of Studies	Years
Berivold et al. [6]	Architecture evolvability analysis	Design-Time	82	1992-2010
Bradbury et al. [S32]	Dynamic software architectures	Run-Time	14	1992-2002
Williams et al. [10]	Architecture change characterization	Both	130	1976-2008
Ahmad et al. [31]	Architecture evolution reuse-knowledge	Both	16	2004-2012
Jamshidi et al. [25]	Architecture-centric software evolution	Both	60	1995-2011

III. Research Methodology

In contrast to a non-structured review process, a systematic review [26, 3] reduces bias and follows a precise and rigorous sequence of methodological steps. It relies on a well-defined and evaluated review protocol [32], outlined in Figure 1. More specifically, we adopted the guidelines in [17] for SLRs with a three step review process that includes: *Planning*, *Conducting* and *Documenting*. The review is complemented with an external evaluation for the outcome of each step, see Figure 1. We also extend the reporting of results so that it provides an explicit taxonomical classification of the reviewed studies. To do so, we take into account the recommended steps on thematic analysis in software engineering [11]. This formulates the foundation for a comparative analysis among studies based on our defined data items that are also subject to external evaluation prior to results reporting [25].

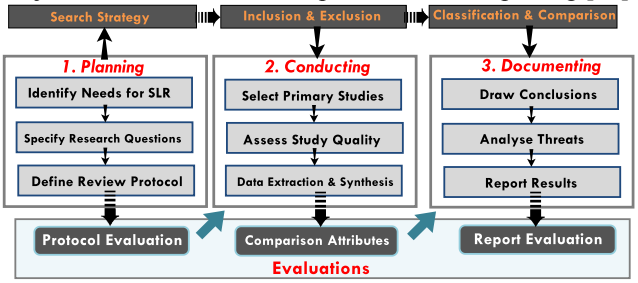


Figure 1. Overview of our research methodology

A. Literature Extraction and Investigation

Four researchers were involved in the literature study. In review planning (*Planning* in Figure 1), a review protocol [32] was defined including the definition of research questions, the search strategy, and initial version of the classification scheme. In terms of search strategy, we combined automatic with manual search. Automatic search was defined as a two-step process for which two categories of search strings (cf. Figure 2) were defined. The first

category selects the studies on architectural constraints which have been *formally* specified, and the second category filters the studies on *architecture-based evolution*. For the manual search, inclusion and exclusion criteria were defined. The classification framework was subsequently defined. The definition of data items was based on information derived from literature sources, specifically the works of [6, 10, 17, S32, 18], and from experience with an earlier review [31]. For some data items, additional attributes were introduced during a pilot run comprising of 10 papers to iteratively evaluate and improve the taxonomical scheme and synchronize understanding of concepts between the researchers. The protocol was cross-checked by an *external* reviewer and the feedback was used to make small adaptations. Subsequently, we conducted the review (*Conducting* in Figure 1). One reviewer was responsible for automated search. Manual search was performed by the other researchers who checked each paper independently based on inclusion/exclusion criteria. Once the primary studies were selected, each study was read by one reviewer to extract the data structured according to the scheme. Collected data items were crosschecked by the other reviewers. Finally, data derived from the primary studies was synthesized, collated, and classified to answer the research questions (*Documenting* in Figure 1).

B. Data Validation and Synthesizing.

When reviewers entered study data into the scheme, they provided a short rationale why the paper should be in a certain category. This rationale is used for *internal* validation purposes. The *external* validation was conducted by an independent researcher outside the working group to provide constructive feedback to the classification scheme and initial review data. The syntheses include the following: (i) classifying and comparing the primary studies, (ii) analyzing of findings and reaching consensus, (iii) interpretation of the results and discussing potential hypotheses for future.

C. ACSE Taxonomical Classification

We utilized a combination of existing ACSE classification and thematic analysis to reduce the time needed in developing the classification scheme. First, the reviewers read abstracts of the 10 selected papers for the pilot run and look for segment of text, keywords and concepts that reflect the contribution of the papers. When this was done, the set of keywords from different papers were labeled, overlaps reduced and combined. This helped the reviewers to define a set of recurrent keywords representative of the underlying population. When abstracts are insufficient to allow meaningful keywords to be chosen, reviewers studied also introduction or conclusion sections. We then clustered the selected set of keywords to create a model of higher-order themes.

IV. An Overview of Planning and Conducting

In this section, we summarize the key steps and outcomes of the *planning* and *conducting* phases of the SLR as illustrated in Figure 1.

A. Research Questions

We formulated the general goal of the study through PICOC (Population, Intervention, Comparison, Outcome and Context) perspectives [22], summarized in Table 2. The central research question translates to five concrete questions:

RQ1: *What types of evolution are supported in ACSE?* The aim is to get insight in what types of evolution are proposed by researchers following four perspectives of evolution: "what", "when", "where", and "why".

RQ2: *Which formalisms are required to enable an ACSE approach?* The aim is to get insight in the usage of formal methods by researchers. This aims to assess which languages and what levels of expressiveness have been used for modeling architectural constraints, verifying properties and automation support.

RQ3: *How architectural reasoning is adopted in ACSE?* The aim is to assess types of constraints and architectural reasoning in existing ACSE.

RQ4: *Which execution environments and mechanisms are needed to enable run-time aspects of ACSE?* The aim is to investigate execution environments or dynamic reconfiguration functionalities used in ACSE.

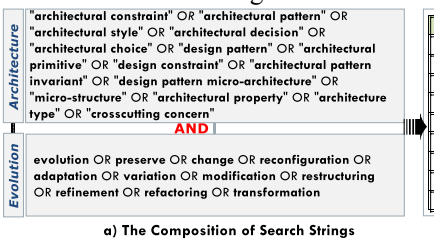
RQ5: *What tool supports is available for ACSE?* The aim is to investigate what automation facilities are utilized to provide support for ACSE concerns.

TABLE 2. PICOC CRITERIA TO DEFINE THE SCOPE AND GOALS OF THIS SLR

Criteria	RQ1	RQ2	RQ3	RQ4	RQ5
Population	Type of Evolution	Type of Specification	Type of Reasoning	Runtime Issues	Tool Support
Intervention	Taxonomical classification; Internal/external validation; Extracting data and Synthesis				
Comparison	A comparison by mapping the primary studies to the classification framework				
Outcome	A classification framework; A comparison framework; Hypotheses for future research				
Context	A systematic investigation to consolidate the peer-reviewed research				

B. Scope of the Study

Once defined the entities and attributes of interest for the framework, the literature extraction process was driven by the following search terms (Figure 2) structured into a logical expression. These search terms are combined by using the Boolean *OR* and *AND* operators that resulted in 14*11=154 search strings in total.



No	Name	Results
1	ACM	663
2	IEEE	1149
3	Science Direct	194
4	Web Of Science	480
5	Springer Link	445
6	Pro-Quest	231
7	Google Scholar	460
8	Wiley	516
	Total	**4138**

b) Search Results on DLs

Figure 2. A summary of search strings and results

After applying the 154 search strings on Google Scholar, IEEE, ACM DL, Springer Link, Science Direct, Wiley Inter-Science, Pro-Quest, and ISI Web of Science, we extracted 4138 manuscripts (Figure 2). Because we used our search criteria on "*title and abstract*", the results provided a relatively high number of irrelevant studies.

Once the initial set of publications had been identified, duplicate publications were removed. These publications were checked against some inclusion and exclusion criteria. Irrelevant publications were removed and this resulted in

235 remaining publications. After further filtering by reading titles and abstracts, 154 publications were left for full text screening to assess their quality [25]. We ranked them accordingly by investigating whether content focuses on formal architecture description or formal analyses related to architectural reasoning. We also checked whether the data analysis of the study is rigorous, based on evidence or theoretical reasoning instead of non-justified or ad-hoc evaluations. Additionally, all references of the 154 studies were checked to ensure no relevant paper was overlooked. Some authors also reported similar results in multiple publications, which we eliminated to avoid bias of the results. At the end, 60 studies were selected as primary studies with the highest ranks. We explicitly defined some general and specific inclusion/exclusion criteria which we described in detail in [32]. We describe the key criteria as described in Table 3.

TABLE 3. INCLUSION AND EXCLUSION CRITERIA

	Criteria	Rationale
Inclusion	**I1**: The study must formalize architecture description as well as architectural properties.	We might have included studies which employ formal methods, but do not actually employ them particularly for evolution.
	I2: The study must provide some evaluation evidences or theoretical reasoning.	We might have included studies which are in their initial stages and mostly are based on trial examples.
Exclusion	**E1**: A study that is an editorial, abstract or a short paper.	These studies do not provide a reasonable amount of information.
	E2: A study that focuses on formal methods themselves, rather than on the use of formal methods for ACSE.	These studies do not provide information regarding the research questions.
	E3: A study that is a review paper.	These studies do not propose any specific approach.

C. Data Extraction and Synthesis

The data extraction process was carried out by reading the 60 papers and extracting relevant data, managed through a bibliographic tool. In order to keep information consistent, data extraction for the 60 studies was driven by the framework (Table 6). The included papers have been initially reviewed and necessary information has been extracted and the framework populated with the extracted information by one of the authors and then cross checked by the other authors. The results of the synthesis will be described in the Section VI.

TABLE 4. STUDY DISTRIBUTION PER PUBLICATION CHANNEL

Publication Channel	Count
Non-Pioneering Software Engineering Journals and Conferences	16
Journal of Systems and Software (JSS)	8
Working IEEE/IFIP Conference on Software Architecture (WICSA)	4
Workshop on Engineering of Computer-Based Systems (ECBS)	4
International Conference on Software Engineering (ICSE)	3
IEEE Transactions on Software Engineering (TSE)	3
Lecture Notes in Computer Science (LNCS)	3
Journal of Information and Software Technology (IST)	3
Software and System Modeling Journal (SOSYM)	2
Electronic Notes in Theoretical Computer Science (ENTCS)	2
Workshop on Sharing and Reusing Architectural Knowledge (SHARK)	2
IEEE International Conference on Software Maintenance (ICSM)	1
International Conference on Quality Software (QSIC)	1
Wiley Journal of Software: Practice and Experience (SPE)	1
European Conference on Software Architecture (ECSA)	1
Springer Software Quality Journal (SQJ)	1
Journal of Visual Languages & Computing (JVLC)	1
Workshop on Self-Managed Systems (WOSS)	1
Workshop on Component-Oriented Programming (CompArch- WCOP)	1
IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE)	1
Future of Software Engineering (FOSE)	1
Total	60

D. Publication Channels and Trends

The selected papers are listed in Table 5. As indicated in Table 4, the majority of these have been published in leading journals and conferences in software engineering in general and software architecture and evolution communities in particular. Column ***A*** in Table 5 shows the counts of sub-criterions which characterize each paper. The studies by year of publication are shown in Figure 3. The trend curve shows a steady number of publications from 1995 to 2003 followed by an increase from 2004 to 2008 and this trend soared in 2010. Note, that for 2011, the review only covers papers until September. While absolute numbers are relatively low, a recent trend indicates a significant increase of publications in ACSE area, especially in 2010 and 2011, which indicates that, as systems are increasingly required to live in an open world [27], the crucial role of architecturally enabled evolution is being recognized [6].

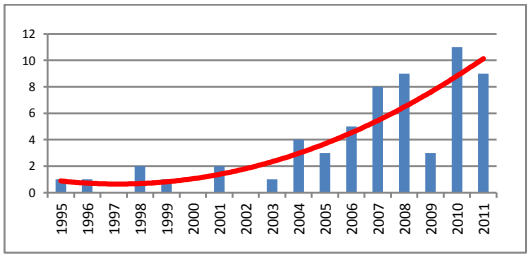


Figure 3. Number of papers by year of publication

TABLE 5. SELECTED PRIMARY STUDIES

Id	Study Title, Corresponding Author	Year	A
S1	XSLT-based evolutions and analyses of design patterns, ***Dong et al.***	2009	17
S2	Behavior-preserving refinement relations between dynamic software architectures, ***Heckel et al.***	2005	18
S3	Combining formal methods and aspects for specifying and enforcing architectural invariants, ***Kallel et al.***	2007	18
S4	Evaluating pattern conformance of UML models a divide-and-conquer approach and case studies, ***Kim et al.***	2008	16
S5	Formal analysis of architectural patterns, ***Caporuscio et al.***	2004	20
S6	Modeling and enforcing invariants of dynamic software architectures, ***Kallel et al.***	2010	19
S7	Style-based modeling and refinement of service-oriented architectures, ***Baresi et al.***	2006	16
S8	A case study in re-engineering to enforce architectural control flow and data sharing, ***Abi-Antoun et al.***	2007	16
S9	A catalog of architectural primitives for modeling architectural patterns, ***Zdun et al.***	2008	17
S10	Automated adaptations to dynamic software architectures by using autonomous agents, ***Wenpin et al.***	2004	17
S11	A family of languages for architecture constraint specification, ***Tibermacine et al.***	2010	20
S12	Architecture compliance checking at run-time, ***Ganesan et al.***	2008	17
S13	A type-centric framework for specifying heterogeneous, large-scale, component-oriented, architectures, ***Jung et al.***	2010	12
S14	Analyzing architectural styles, ***Kim et al.***	2010	12
S15	Changing attitudes towards the generation of architectural models, ***Castro et al.***	2011	12
S16	Classifying architectural constraints as a basis for software quality assessment, ***Giesecke et al.***	2007	4
S17	A rule-based method to match software patterns against UML models, ***Ballis et al.***	2007	17
S18	Deriving detailed design models from an aspect-oriented ADL using MDD, ***Pinto et al.***	2011	12
S19	Formal specification of the variants and behavioral features of design patterns, ***Bayley et al.***	2010	12
S20	Model-driven development for early aspects, ***Sánchez et al.***	2010	14
S21	Pattern-based design evolution using graph transformation, ***Zhao et al.***	2007	16
S22	PS-CoM building correct by design Publish Subscribe architectural styles with safe reconfiguration, ***Loulou e al.***	2010	14
S23	Understanding the relevance of micro-structures for design patterns detection, ***Arcelli et al.***	2011	9
S24	Using aspects for enforcing formal architectural invariants, ***Kallel et al.***	2008	14
S25	Evolution styles to the rescue of architectural evolution knowledge, ***Le Goaer et al.***	2008	16
S26	A model transformation approach for design pattern evolutions, ***Dong et al.***	2006	13
S27	A scalable approach to multi-style architectural modeling and verification, ***Wong et al.***	2008	10
S28	A UML rule-based approach for describing and checking dynamic software architectures, ***Miladi et al.***	2008	14
S29	Correct architecture refinement, ***Moriconi et al.***	1995	12
S30	Preserving Architectural Choices throughout the Component-based Software Development Process, ***Tibermacine et al.***	2005	15
S31	Style-based refinement of dynamic software architectures, ***Baresi et al.***	2004	17
S32	A survey of self-management in dynamic software architecture specifications, ***Bradbury et al.***	2004	20
S33	A contract place where architectural design and code meet together, ***Ubayashi et al.***	2010	14
S34	Design pattern solutions as explicit entities in component-based software development, ***Stepan et al.***	2011	11
S35	Modeling architectural patterns using architectural primitives, ***Zdun et al.***	2005	15
S36	Simplifying transformation of software architecture constraints, ***Tibermacine et al.***	2006	14
S37	A constraint architectural description approach to self-organizing component-based software systems, ***Waewsawangwong et al.***	2004	16
S38	A constraint-oriented approach to software architecture design, ***Van den Berg et al.***	2009	8
S39	A framework to specify incremental software architecture transformations, ***Barais et al.***	2005	17
S40	An approach based on biographical reactive systems to check architectural instance conforming to its style, ***Chang et al.***	2007	16
S41	An automated refactoring approach to design pattern-based program transformations in Java programs, ***Jeon et al.***	2002	14
S42	Analyzing and comparing architectural styles, ***Levy et al.***	1999	14
S43	Architecting as Decision Making with Patterns and Primitives, ***Zdun et al.***	2008	17
S44	Architecture-based runtime software evolution, ***Oreizy et al.***	1998	12
S45	Capturing interactions in architectural patterns, ***Yadav et al.***	2010	14
S46	Describing evolving dependable systems using co-operative software architectures, ***de Lemos et al.***	2001	18
S47	Describing software architecture styles using graph grammars, ***Le Metayer et al.***	1998	15
S48	Design pattern evolution and verification using graph transformation, ***Zhao et al.***	2007	18
S49	Evaluation of accuracy in design pattern occurrence detection, ***Pettersson et al.***	2010	12
S50	Evolution styles foundations and tool support for software architecture evolution, ***Garlan et al.***	2006	17
S51	Focus: a light-weight, incremental approach to software architecture recovery and evolution, ***Ding et al.***	2001	14
S52	Formal specification of design patterns and their instances, ***Taibi et al.***	2006	14
S53	Guiding architectural restructuring through architectural styles, ***Tamzalit et al.***	2010	17
S54	Safe integration of new concerns in a software architecture, ***Barais et al.***	2006	16
S55	Scenario-based architectural design decisions documentation and evolution, ***Che et al.***	2011	13
S56	Self-managed systems an architectural challenge, ***Kramer et al.***	2007	13
S57	Style-based reuse for software architectures, ***Monroe et al.***	1996	11
S58	Synthesizing approach for perspective based architecture design, ***Liang et al.***	2003	10
S59	Towards a formal model for reconfigurable software architectures by bi-graphs, ***Chang et al.***	2008	15
S60	Using UML2.0 and GG for describing the dynamic of software architectures, ***Kacem et al.***	2005	18

V. The Classification Framework

In this section, we propose a *model* consisting of 21 analysis dimensions to *characterize* ACSE approaches. This model constitutes a foundation for evaluating ACSE approaches that leads to a classification and comparison of selected studies comprising those approaches. For each of the analysis dimensions (18 technical and 3 non-technical), the model considers a set of standardized characterization options. These options resulted from combining classification attributes from recognized sources with those found in the set of papers that we selected. For instance, the set of options for the "Need for Evolution" was identified using the ISO/IEC 14764 Standard for Software Maintenance and, on the other hand, "Type of Formalism" and "Description Language" were both mainly identified using [S32] and [17] respectively. They, however, have been improved over time by considering the new options in the selected studies. As a driver for the selection of the model parameters, Section A provides a rationale for the proposed classification, while Section B outlines the details of various entities and attributes as a fine granular representation of the classification framework.

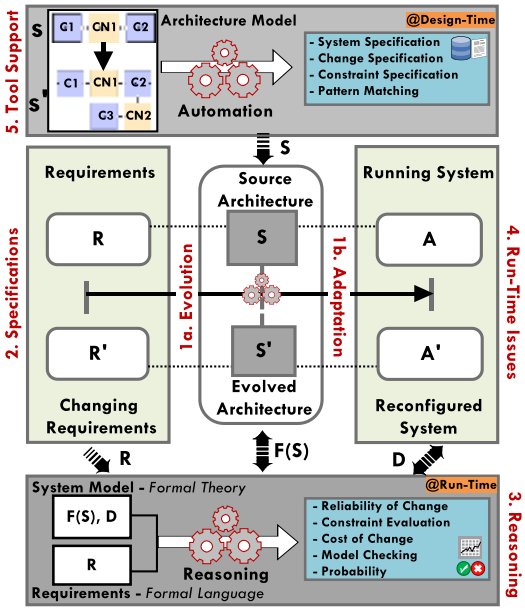


Figure 4. The conceptual framework

A. Rationale for the Classification

A *causally connected* architecture model, which is denoted by ***S,*** conceptually represents a running software system ***A*** embedded in an execution environment. Now let ***D*** be the domain assumptions that are consistent with the ***S*** (Figure 4). Then it should be proved that they hold the requirements ***R***. This can be *formally* expressed as $F(S), D \vDash R$, where **F** is a function that maps the architecture specification into a semantic model. Software evolution deals with the *violation of correctness* criteria after ***A*** is embedded in the environment and starts to operate [24]. The violation may occur as a result of the divergence of (i) the system ***A*** from specification ***S***, (ii) the environment behavior from specification ***D***, (iii) the system goals from requirements ***R***. Usually, a software system is taken down to apply a patch offline during maintenance [17]. However, this scenario cannot satisfy the requirements of a specific class of software systems (e.g. mission-critical or safety-critical or business-critical) in which the system must operate continuously and remain capable of on-the-fly changes to the *running* system as a result of the violation $F(S), D \nvDash R$ [S32]. Based on the classification in the context of Figure 4, we are interested in the approaches which deal with *reasoning* about *evolutionary* aspects enabled by formal methods. Software architecture models therefore provide the required abstraction and facilitate the reasoning about the evolution of a system expressed by a formal model (e.g. Markov model) [33]. We consider *ACSE* as a collection of operational and analytical activities to evolve a software system from an older version to a newer version, enabled by architecture changes. Among approaches which use architecture description to operationalize evolution [1], we are interested in approaches which utilize *formal theory* to enable *analytical reasoning* to verify the system properties.

B. The Framework Entities and Attributes

Based on that discussion we understand that ACSE has different dimensions. As conceptually outlined in Figure 4, the architecture of the system needs to be *specified* (**Type of Specification**). For analytical purposes, the properties of systems are also specified as a part of the architecture description as a number of *constraints* (**Type of Reasoning**). An *evolution* mechanism (**Type of Evolution**) can analyze consequences and apply changes either at design-time or *runtime* (**Run-time Issues**). For scalability, performance and economic issues, these activities require automation (**Tool Support**). As far as the mentioned goals are concerned, we propose the classification scheme as presented in Table 6.

Table 6. The ACSE classification framework

Taxonomical Classification (Higher-Order Theme)	Sub-classifications (Sub-themes)	Coded Attributes (Domain Knowledge, Standards, Keywords)
Type of Evolution	Need for Evolution	***ISO/IEC 14764 typology of change***: Corrective, Perfective, Adaptive, Preventive, All applicable
	Means of Evolution	***Static***: Transformation, Refactoring, Refinement, Restructuring, Pattern change; ***Dynamic***: Reconfiguration, Adaptation
	Time of Evolution	Design-Time, Run-Time
	Support Activity	***Change impact***: Consistency checking, Impact analysis, Propagation; ***Change history***: Evolution analysis, Versioning
	Stage of Evolution	***SDLC***: Analysis/Design, Implementation, Integration/provisioning, Deployment, Evolution
Type of Specification	Level of Formalism	Lightweight, Formal
	Type of Formalism	***Modeling language***: ADL, Programming lang., Domain-specific lang., Type systems, Archface, Model-based; ***Formal models***: Graph theory, Petri-net, Ontology, State machine, Constraint automata, CHAM; ***Process algebra***: FSP, CSP, π-calculus; ***Logic (Constraint language)***: OCL, CCL, FOL, Grammars, Temporal logics, Rules, Description logic, Z, Alloy, Larch
	Description Language	***Process algebra***: Darwin, Wright, LEDA, PiLar; ***Standards***: UML, Ex.-UML, SysML, AADL; ***Others***: ACME, Aesop, C2, MetaH, Rapide, SADL, UniCon, Weaves, Koala, xADL, ADML, AO-ADL, xAcme
	UML Specification	***Static***: Class, Component, Object; ***Dynamic***: Activity, State, Sequence, Transition, Communication
	Description Aspect	Structural, Behavioral, Semantic
Type of Reasoning	Architectural Constraint	***Structural***: Pattern, Architectural style, Primitive, Metaphore, Micro-structure, Crosscutting concern, Clue; ***Invariant***: Component-level invariant, Cross-component invariant, Pre-/Post-condition, Temporal invariant; ***Relational constraints***: Coding rules, Cardinality, Coordination, Meta-model, Variability
	Intent of Reasoning	Specify, Preserve, Change, Enforce, Match, Discover, Analyze
	Type of Analysis	Consistency checking, Model checking, Pattern conformance, Graph refinement, Enforcement
Run-Time Issue	Runtime Environment	***Component model***: Fractal, KOALA, SOFA 2.0, CCM, OpenCOM, KobrA; ***Middleware***: SIENA, OSGi, .NET, MS-COM, EJB, JavaBeans, SOA middleware, ***Coordination model***: Reo, Linda, MANIFOLD
	Mechanism of Runtime Evolution	***Reflection mechanisms***: Introspection, Constraint injection, Intercession, Reification, Causal connection, ***Evolution enablers***: Safe stopping, Runtime binding, State transfer
Tool Support	Need for Tool Support	**Architecture life-cycle**: Business case, Creating architecture, Documenting, Analyzing, Evolving
	Analysis Usage of Tool Support	Simulation, Dependence analysis, Model checking, Conformance testing, Interface consistency, Inspection and review-based
	Level of Automation	Fully automated, Partially automated, Manual
Research Method	Research Motivation	A particular problem/challenge, Overview or Survey, Formalism for constraint specification, Formalism for architectural analysis, Formalism for arch. evolution, Formalism for code generation
	Application Domain	***Development paradigm***: SPL, OO, SOA, CBS; ***Traditional***: Embedded, Real-time, Process-aware, Distributed, Event-based, Concurrent, Mechatronic, Mobile, Robotic, Grid computing; ***Emerging***: Cloud computing, Smart-*, Autonomic computing, *-critical, Ubiquitous
	Evaluation Method	Case study, Mathematical proof, Example application, Industrial validation

The groupings and their dimensions are shown in Table 6. Note that the proposed framework is used for the comparison of ACSE approaches based on our focus which was discussed in the previous section and should not be adopted without a justification or verification. Firstly, we deliberately did not include all aspects of software

architecture evolution. For example, the *who* question, which identifies the stakeholders involved in software architecture change, is not covered in the current classification because it reflects human aspects of evolution and our focus of rigorous formal theory, analytical reasoning based on architectural constraints and runtime issues aims to lessen human intervention. The *where* question is also excluded because it has been thoroughly covered in [17] for specification-time evolution and in [S32] for runtime reconfiguration. Secondly, the proposed framework provides only one of the many ways in which ACSE can be characterized. Most importantly, the framework is subject to continuous improvements, since the comparison attributes continue to mature due to scientific and technological advances. A list of improvement suggestions can be found in the protocol [32].

VI. Results and Comparative Analysis

We compare ACSE approaches along the dimensions summarized in Table 6. The results of this analysis process are summarized in Tables 7-12 below. Table 7-11 presents the technical characterization of selected approaches based on the model proposed in Section V. Table 12 classifies the studies based on their research method. Column ***P*** shows the percentage of studies classified in each sub-criterion. The last column shows the percentage of frequencies in three yearly disjoint categories (1995-2005, 2006-2008, and 2009-2011) for attributes that characterize 10 or more approaches.

A synthesized analysis of the studies is discussed in each section and the investigated approaches are critically summarized in subsequent tables. Mapping the studies to the common scheme provides us with a unified framework to group contributory artifacts of similar approaches. These categories provide a means to compare ACSE approaches. Moreover, the analysis of evidence based on a considerable amount of research or lack of focus on specific attributes, differentiated by the count column in the subsequent tables, are the critical discussions of the next subsections - structured by the sub-criteria. Each study may be represented by multiple attributes in case of non-disjunct attributes. Due to the limited amount of data, a statistical analysis was not feasible. Thus, we limit the discussions to results based on the table content, deriving future directions in each area using quantitative summaries of the data. This approach usually characterizes systematic mapping studies [11]. At the same time, whilst addressing the reliability of the included studies, the research and evaluation method of the approaches were also synthesized. This comparison based on a common scheme reveals that e.g. a majority of studies focuses on specific aspects of evolution or there is a lack of studies that point out gaps.

A. Type of Evolution

This category compares the included studies in terms of a general taxonomy which focuses on the factors that characterize and position these mechanisms into a general purpose classification. The studies are further classified into five sub-categories explained below, summarized in Table 7.

(i) Need for evolution. According to the ISO/IEC 14764 standard for software maintenance and Lehman's law of software evolution [34], the *reason for evolution* (purpose of change) can be categorized into four types: corrective, perfective, adaptive and preventive. Most approaches are applicable for all purposes, but among them *adaptive changes* in the context of runtime evolution [S44] was explicitly mentioned in considerable number of studies. **(ii) Means of evolution**. Among the different types of evolution, *transformation*, *reconfiguration*, and *pattern-based evolution* gained significant attention. Refinement, restructuring, adaptation and architectural decision evolution were explicitly adopted as a promising mechanism for enabling evolution in the studies. **(iii) Time of evolution**. *Specification-time* software architecture evolution has been more popular than *runtime* evolution. The relative lack of contributions on runtime evolution results from a recent adoption of reflective component models that provide architecture information to the mechanisms that enable the dynamic reconfiguration. However, dynamic evolution recently gained a momentum. **(iv) Support activities**. ACSE related activities contain analytic work to check relevant properties of architectures after the evolution takes place. For instance, consistency checking aims to check whether the system architecture is consistent after the requested changes. *Consistency checking* is a popular architecture analysis, but other context-based analyses were also utilized. **(v) Stage of evolution**. The focal stage of the software lifecycle was considered. Both early and late stages of the lifecycle were key areas of attention in comparison. *Design* and *maintenance* stages were the two stages in which evolution mechanisms were active.

Table 7. Comparative Analysis Based on "Type of Evolution"

Sub-Criteria	Attributes	Approaches	P%	N	Distribution % 95-05	06-08	09-11
Need for Evolution	Corrective	[50]	52	1			
	Perfective	[12]		1			
	Adaptive	[1,9,40,42,46,50,53,56,59,60]		10	**30**	**50**	**20**
	Preventive	[45,46,52]		3			
	All applicable	[2,3,5,6,8,10,11,17,37,39,41,44,47,48,49,51,54,55]		18	**50**	**28**	**22**
Means of Evolution	Transformation	[1,2,6,7,8,21,26,28,29,31,33,36,39,40,50,58]	83	16	**31**	**50**	**19**
	Refactoring	[8,41,51]		3			
	Refinement	[2,7,13,22,29,31,43,51,58]		9			
	Restructuring	[8,13,50,53]		4			
	Adaptation	[4,6,10,44,46,56]		6			
	Reconfiguration	[2,3,4,5,6,7,22,24,28,31,32,37,54,59,60]		15	**40**	**47**	**13**
	Pattern-based	[1,9,17,21,26,34,35,41,42,45,48,49,52]		13	**23**	**46**	**31**
	Arch. decision	[11,12,25,47,55,57]		6			
	Change operation	[4, 28]		2			
Time of Evolution	Design-Time	[1,2,4,7,9,11,13,14,15,17,18-23,25-30,33-36,39,41-43,45-55,57,58]	95	43	**27**	**35**	**37**
	Run-Time	[2,3,5-10,12,24,31,32,37,40,44,54,56,59,60]		19	**42**	**53**	**5**
Support Activity	Consistency checking	[2,11,33,43,48,60]	18	6			
	Evolution analysis	[5,7]		2			
	Change propagation	[3,6]		2			
	Versioning	[50]		1			
	Impact analysis			0			
Stage of Evolution	Analysis and Design	[1,4,6,8,9,13-16,18-20,22,25,26,27,29,30,33-36,38,39,42,43,45,47,48,52,57,58]	92	32	**25**	**34**	**31**
	Implementation	[3,6,8,24,33,34]		6			
	Evolution	[3,5,6,10,12,17,21,23,28,31,37,40,41,44,46,49,50,51,53,54,56,59,60]		23	**39**	**44**	**17**
	Integration and provisioning, Deployment			0			

As a concluding remark, the recent boost in autonomic computing [7] and specifically self-adaptive applications [29] also reflects that *reliable evolution* of running software has become a major challenge. This trend which is enabled mostly by architecture-based runtime software evolution [S44] is expected to get more attention based on the current publication trend on runtime evolution. Software systems in open worlds [27] necessitate enabling *run-time adaptability* for systems to cope with environmental uncertainties, changing requirements, and failures.

B. Degree of Formalism and Expressiveness

This category compares the included studies in terms of the degree of expressiveness and analyzability employed in ACSE approaches. The design of languages and formalisms involves a trade-off between expressive power and analyzability. The more formalism can express, the harder it becomes to understand what instances of the formalism say [12]. The degree of formalism is due to the ACSE nature that requires utilizing analytical power of description languages, e.g., to check consistency or integrity after evolution or derive impact of primary changes or quantitatively verify runtime changes for dynamic adaptations. The language aspect has been exhaustively investigated in a comprehensive framework [17]. Therefore, we briefly review our observation in terms of the attention of ACSE studies to the language aspect. Based on this theme, the studies are further classified into five sub-categories explained below, summarized in Table 8.

TABLE 8. COMPARATIVE ANALYSIS BASED ON "TYPE OF SPECIFICATION"

Sub-Criteria	Attributes	Approaches	P%	N	Distribution %		
					95-05	06-08	09-11
Level of Formalism	Lightweight	[1,8,9,12,13,15,18,20,23,25,26,33-35,36,38,55]	88	17	8	46	46
	Formal	[2-7,10,11,14,17,19,21,22,24,27-32,37,39-43,45-48,50,52-54,59,60]		36	38	42	20
Type of Formalism	Graph	[2,6,7,21,22,31,32,40,47,48,53,59,60]	83	13	38	39	23
	Petri-net	[3,12,24]		3			
	Model-based	[1,2,4,5,6,7,9,11,13,15,18,19,20,26,28,35,55,56,60]		19	21	32	47
	Description logic	[29,32,41,52]		4			
	π-calculus	[10,45]		2			
	Prog. language	[8,3,17,41]		4			
	Alloy	[14,27,37]		3			
	OCL	[2,4,6,7,9,11,25,28,30,31,35,36,43,60]		14	36	50	14
	CCL	[11,36]		2			
	State machine	[5]		1			
	Z	[3,6,22,24,28]		5			
	C. Automata	[39,46]		2			
	Process algebra	[32,42]		2			
	Temporal logics	[5,19,50,52]		4			
	Archface	[33]		1			
	Ontology, Domain-Specific, Larch, Grammars, Type, FSP, CSP, CHAM, FOL, Rules			0			
Description Language	ACME	[8,14,15,25,30,36,46,50]	58	7			
	xAcme	[30,36]		2			
	UML	[1,5,17-21,26,30,31,33, 48,51,53,56,60]		16	31	31	38
	Extended UML	[2,4,6,7,9,11,25,28,35,39,43]		11	27	55	18
	AO-ADL	[18]		1			
	Darwin	[37,56]		2			
	SafArchie	[39,54]		2			
	Aesop, SADL, Koala, xADL, ADML, AO-ADL, UniCon, Weaves, Wright, C2, Rapide, MetaH, AADL			0			
UML Specification	Activity	[2,28,43,53]	55	4			
	State	[2,5]		2			
	Sequence	[5,7,18,19,20,51]		6			
	Class	[1,4,7,8,9,11,17,19,21,25,26,31,33,35,39,48,51,52,60]		19	26	53	21
	Component	[2,6,7,8,18,20,28,30,35,43,46,50,51,53,54,57,60]		17	41	35	24
	Object	[2]		1			
	Communication	[31,33]		2			
	Transition	[2,3]		2			
Description Aspect	Structural	[1-4,6-15,17,19-22,24-30,32-34,36,37,39-57,59,60]	93	52	28	42	30
	Behavioral	[2,3,5,7,9,10,12,19,20,22,24,27,29,31-35,39-42,44-46,49-57]		34	41	32	27
	Semantic	[43,58]		2			

(i) Level of formalism. The main observation, which is biased because of our inclusion/exclusion criteria, is that formal approaches dominate the ACSE area. **(ii) Type of formalism.** Traditional modeling languages such as graph and model-based languages are the most popular. The majority of studies formulate the architectural constraints corresponding to some properties of interest in OCL or a variation of logic as constraint specification language. We observed that the majority of the studies either formulate the properties of interest in the modeling language or they utilize a logic combined with a modeling language. Recently, the creation of new languages has declined and researchers tend to combine or extend existing languages for their own purposes. **(iii) Description language.** UML extensions [12], especially component and class diagrams (**(iv) UML specification**), are widely adopted by ACSE approaches. Although UML has restricted architectural analysis support, its wide adoption is based on its support for different domains and business contexts [12] and its standardization. Other description languages such as ACME, Darwin or SafArchie are approximately equally distributed. **(v) Description aspect**. We observed that both structural and behavioral aspects have been addressed. However, semantic aspects have been neglected.

We observed advances in the fundamentals of architecture modeling, but a unifying formalism that provides the basis for all architecture analysis requirements is unlikely [12]. Constraint specification languages are growing due to the need to specify architectural properties that need to be verified against a system model (Figure 4). The coverage of topics points out a gap regarding formalisms addressing architecture-centric evolution augmented with more *semantic enabled* reasoning. This is particularly challenging for specifying the system model and reasoning based on a richer semantic model at runtime. Therefore, description languages need more integration with underlying formal *theories* to enable specification of emerging *architectural properties*.

C. Type of Architectural Reasoning

This category compares the included studies in terms of constraints [S16] on architecture description to enable *automated reasoning*. Architectural constraints are an inherent part of ACSE approaches since they facilitate evaluating system properties. The studies are classified into three sub-categories, see Table 9.

TABLE 9. COMPARATIVE ANALYSIS BASED ON "TYPE OF REASONING"

Sub-Criteria	Attributes	Approaches	P%	N	Distribution %		
					95-05	06-08	09-11
Architectural Constraint	Pattern	[1,5,11,16,17,19,21,23,26,34,35,39,41-43,45,48,49,52,57]	85	20	30	35	35
	Architectural style	[7,10,12,25,36,37,38,40,42,43,44,46,47,50,51,53,57-60]		20	50	40	10
	Primitive	[1,9,26,35,39,41,43,54]		8			
	Component-level invariant	[30,39,60]		3			
	Cross-component invariant	[2,3,9,20,25,30,36,39,42,44,50,56,58,60]		14	50	43	7
	Crosscutting concern	[18,39]		2			
	Meta-model	[4,7,9,43]		4			
	Pre-/Post-condition	[2,3,4,6,7,22,24,25,31,39,46,55,56,60]		14	36	43	21
	Coordination constraint	[3,24]		2			
	Coding rules	[11]		1			
	Cardinality, Metaphore, Micro-structure, Clue, Variability, Temporal			0			
Intent of Reasoning	Specify	[5,9,11,13,14,15,24,25,27,28,35-37,39,43,45,47,50-56,58-60]	97	27	30	44	26
	Preserve	[1,2,6-11,18-22,25,26,29-31,35,36,40,41,44,47,48,50,51,53-60]		35	37	37	26
	Change	[11,39,40,41,46,51,60]		7			
	Enforce	[3,6,8,11,13,24,33,34,46,50]		10	10	40	50
	Match	[4,12,17,40,42,49,50]		7			
	Discover	[17,23,38,49]		4			
	Analyze	[3,5,6,11,14]		5			
Type of Analysis	Consistency checking	[1,6,9,10,11,12,21,23,30,39,40,46,48,60]	55	14	36	36	28
	Model checking	[2,3,5,22,24,27,29,31,32,37]		10	60	30	10
	Pattern conformance	[4,17,41,54]		4			
	Graph-based refinement	[7,53]		2			
	Constraint enforcement	[8,25,59]		5			

(i) Means of constraint. Architectural styles, patterns, component-level invariants, cross component invariants, pre- and post-conditions which have an explicit description separable from the architecture of a system under consideration, are determinants of some characteristics of the system [13]. **(ii) Intent of constraint.** Specification, preservation and enforcement of constraints are the main reasons behind constraints in the approaches. **(iii) Analysis of constraint**. Most approaches are accompanied or enabled by analytical mechanisms such as consistency checking, model checking, and conformance checking.

We observed a trend towards *quantitative verification of system properties* based on constraints for correct adaptations, especially in self-adaptive systems which operate in uncertain environments that promote the use of probabilistic modeling of systems.

D. Run-time Issues

This category compares the studies in terms of underlying environment and mechanism to enable dynamic reconfiguration. It identifies the prerequisites for what is

needed to implement and utilize the often complex and hypothetical conceptual contributions in the context of reliable runtime infrastructures. The studies are classified into two sub-categories (Table 10).

(i) Environment of runtime evolution. A variety of execution environments has been proposed, ranging from traditional platforms such as CORBA Component Model (CCM) to the reflective and flexible Fractal and OpenCOM to a new SOA and Cloud based platforms. **(ii) Mean of runtime evolution**. To enable runtime evolution, these environments need to expose functions to provide the current state of a system at a specific time during its execution and a causal connection between the running system and architectural information. Ideally, they provide facilities to change architectural descriptions at runtime.

TABLE 10. COMPARATIVE ANALYSIS BASED ON "RUNTIME ISSUES"

Sub-Criteria	Attributes	Approaches	P%	N
Environment of Run-Time Evolution	Fractal	[30]	10	1
	OSGi	[12]		1
	CCM	[11]		1
	PKUAS	[10]		1
	SOA middleware	[7]		1
	SIENA	[5]		1
	KOALA, MS COM, SOFA 2.0, EJB, JavaBeans, OpenCOM, KobrA, .NET, Reo, Linda, MANIFOLD			0
Mechanism of Run-Time Evolution	Reflection	[10]	8	1
	Causal connection	[5,9,11]		3
	Runtime binding	[7]		1
	Introspection, Constraint injection, State transfer, Intercession, Reification, Safe stopping			0

There is a lack of approaches for *runtime environments* and *evolution*. We suppose this is because implementations of evolution mechanisms on top of these technologies are not feasible in research domains and researchers prefer to rather provide evidence or theoretically prove their contributions. Recent efforts identified an extensive list of run-time concerns – mainly by autonomic computing communities [29]. Disciplined engineering approaches to support decentralization and making contextual system models accessible and machine process-able at runtime [14] are still relevant concerns [29].

E. Tool Support

This category compares the included studies in terms of infrastructure which support automation of ACSE. This theme provides a narrow view of automation for ACSE. Based on this, the studies are classified into three sub-categories (Table 11).

TABLE 11. COMPARATIVE ANALYSIS BASED ON "TOOL SUPPORT"

Sub-Criteria	Attributes	Approaches	P%	N	Distribution % 95-05	06-08	09-11
Need for Tool Support	Creating	[15,20,22,27,29,30,34,38,45]	95	9			
	Documenting	[1,3,8,9,11,35,36,43,52,55,57]		11	**18**	**55**	**27**
	Analyzing	[1-3,5,6,8,12-14,17,23,33,38,49]		14	**14**	**29**	**57**
	Evolving	[3-7,9-11,18,21,24,25,28,31-33,36,37,39-42,44, 46-48,50,51,53-56,58-60]		35	**40**	**43**	**17**
	Business case			0			
Analysis Usage of Tool Support	Simulation	[7,31]	30	2			
	Dependence analysis	[5,11]		2			
	Model checking	[2,3,5,6,7,14,31,33,43,46]		10	**40**	**30**	**30**
	Conformance	[4,10,12,15,40,47]		6			
	Inspection	[20]		1			
	Interface consistency			0			
Level of Automation	Full	[4,21,30,41,54]	100	5			
	Partial	[1-3,5-12,14,17,18,20,22,24,26-29,31,33,39,40, 43,44,48,49,50,51,53,57,58]		34	**29**	**41**	**30**
	Manual	[13,15,16,19,23,25,32,34-38,42,45-47,52,55,56,59,60]		21	**33**	**29**	**38**

(i) Need for tool support. Although most of the studies use tools for architectural evolution, the trend shows that employing tools for analysis purposes is gaining significant attention. **(ii) Analysis usage of tool support.** The studies that use tools employ them mostly for model checking. **(iii)Level of automation.** Most of the studies are either fully automated for both architecture evolution and analysis or partially automated in either of them. Thus, unsurprisingly, the majority of studies use tool support either for *proof of concept* or *automated analysis* purposes.

F. Research methodology of the studies

This category differs from the other 5 purely technical aspects and only reflects general aspects (Table 12).

TABLE 12. COMPARATIVE ANALYSIS BASED ON "RESEARCH METHOD"

Sub-Criteria	Attributes	Approaches	P%	N	Distribution % 95-05	06-08	09-11
Motivation	Problem or Challenge	[2,3,6,7,15,18,20,25,38,42,49,50,52,54,56-58]	98	17	**24**	**41**	**35**
	Overview or Survey	[8,16,32,38,49]		5			
	Unambiguous description	[4,9,11,14,19,22,26,27,30,35,45,47,52]		13	**23**	**38**	**39**
	Analysis	[4,5,9,12,14,17,23,28,33,35,40,49,55]		13	**15**	**46**	**39**
	Evolution	[1,6,10,13,18,20-22,24,26,28,29,31,33,34,36,37,39-41,44, 46-48,50,51,53,55,59,60]		30	**37**	**30**	**33**
	A formalism to enable code generation			0			
Application Domain	SOA	[1,2,7,31,43]	78	5			
	OO	[1,3,8,17,21,26,41,42,48,49,51,57]		12	**33**	**50**	**17**
	Distributed system	[3,24,46,60]		4			
	Event-based system	[5,22,44]		3			
	Component-based	[6,10-13,22,24,25,28,30-37, 39,40,43,45-47,50,53-60]		32	**38**	**34**	**28**
	SPL, Embedded, Ubiquitous, Mission-critical, Real-time, Process-aware, Concurrent, Mechatronic, Mobile, Robotic, Cloud computing, Smart-*, Autonomic computing, Grid computing			0			
Evaluation Method	Case study	[1,4-9,12,14,15,18-20,22,24,29,35,37,38,40,43,46,47,51, 53,55,60]	87	27	**29**	**30**	**41**
	Mathematical proof	[1-3,5,17,19,21,22,29,47,54]		11	**36**	**36**	**28**
	Example application	[3,11,13,14,21,25,27-31,33,34,36,37,39,41,42,44,45,48-50,52,54,59]		26	**31**	**42**	**27**
	Experience report			0			

(i) Motivation. The main focus of research is on formalisms with attention to enabling evolution in software systems and enabling automated mechanisms to support evolution. **(ii) Application domain**. The dominant application domain is component software, accounting for more than 50% of applications. This portion is increasing. Services have also gained attention recently. These two domains require dynamic reconfiguration. In particular, dynamic components and service reconfiguration are active areas of research, with particular attention for rigorous approaches to support challenges in self-adaptive systems and dynamic selection and composition of elements. Furthermore, dynamic domains such as robotics and smart cities have gained little attention. **(iii) Evaluation method**. Although most approaches included some formal proofs as part of the evaluation, evidence is often obtained from applying the approach to small case studies and examples. Only a few empirical studies have been undertaken; no industrial evidence is reported. Lack of rigorous validation and attention to application domains were major concerns that we found in the included studies. These concerns and domains offer areas for future research on ACSE.

VII. FUTURE RESEARCH DIRECTIONS AND LIMITATIONS

Based on the taxonomical classification and holistic comparison, we summarize research implications and future directions. We also discuss possible limitations as well as threats to validity of this SLR.

A. Research Implications

This classification and comparison framework has implications for researchers and practitioners as summarized with the ACSE dimensions in Table 12. For *researchers*, the classification framework provides a comprehensive view of different aspects to be considered when addressing an ACSE problem. The comparison of existing work highlighted several areas where extensive support is provided or gaps are pointed out, see Table 13.

Implications for future research. Based on the systematic consolidation, we identified a number of areas that lack a more rigorous research as emerging trends in ACSE. The list (Table 13) is not exhaustive, but reflects

key challenges identified in the consolidation and corresponds to the classification in Section VI.

(i) **Evolution**: In the context of more classical Lehman's law [34], we observe ever growing needs for adaptive evolution with primary focus on run-time architectural adaptation. An interesting observation is the emergence of '*adaptation pattern*' [31] to support reuse in evolution for dynamic adaptive software architecture. *Mining software repositories* to empirically analyze changes and fostering them for reuse is also getting more attention. Researchers look at long running, large applications such as Eclipse SDK [15] in order to further this domain by learning from past recurring evolutions.

(iii) **Architectural reasoning**: A considerable number of approaches address *probabilistic modeling* of systems and their environments which they operate and also explicit specification of *architectural constraints*. Analytic techniques verify specific properties or preserve the initial constraints. However, these techniques that are normally conceived as design-time activities must extend their scope to runtime and comply with the time constraints of runtime environment. We observe that the need for *runtime verification and validation* to detect violations and plan self-reactions demands for *efficient* analytical and syntactical "@runtime" techniques [2, 24].

(iv) **Run-Time evolution**: In a considerable number of approaches, *formal specifications* are applied during development activities for verification and validation, rather than the system itself at runtime. More importantly, those approaches that utilize formal specification at runtime use it for mainly modeling and analysis rather than other ACSE activities. We observe an improvement of adaptive middleware based on *reflection* mechanisms. However, *decentralized* evolution and *@runtime* notions are still relevant concerns for further research in ACSE.

(vi) **Evaluation evidence**: An important issue that we observed and is asserted by work by Barnes at NASA JPL regarding space systems architectural evolution is the *lack of rigorous validation* [21]. Previous work tended to make heavy use of trial examples and unrealistic assumptions and has not been well supported by real-world scenarios. We came across this when excluding a large number of studies that had been retrieved based on our search, but violated an inclusion criterion (I2 in Table 3). To stimulate ACSE community to produce empirical real-world case studies, such efforts need recognition at established conferences [7].

Implications for practitioners. The data [25] collected may help *practitioners* in searching for relevant approaches before adopting and tailoring them by examining the context of their own software development, and comparing with characteristics of relevant approaches. However, [28] provides an overview of various approaches evaluated based on real-world industrial scenarios concerning evolution of sustainable industrial systems. We believe that this document can suitably assist practitioners because it is more general based on a growing number of experience reports which might not necessarily be peer-reviewed.

Practical application of the collected database. Since this classification framework and its accompanying templates contain more than 160 attributes, it provides a large amount of information. For instance, for the 60 papers, it makes a collection with 160*60=9600 data points which are quite significant. As a result, the user can, for example, query the database to find *case studies* on *industrial applications* of *petri-net* based approaches for *runtime evolution*. Moreover, this database can grow and be used as a knowledge-base for ACSE researchers, such as [28] for practitioners.

TABLE 13. RESEARCH DIMENSIONS AND FUTURE IMPLICATIONS FOR ACSE

Research Needs / Dimensions	Evolution	Formalism	Architectural Reasoning	Run-Time Issues	Evaluation
Research Trend	Self-adaptive systems	Emerging properties	Quantitative verification	Decentralized evolution	Rigorous validation
Active Topics	Change patterns, Repository mining	Formal theory	Explicit constraints, Probabilistic modelling	Model@run-time, Reflective environment	Empirical studies
Targets	Self-reaction to violations of constraints	Semantic models	Constraint preservation, property verification	Efficiency and Reliability in evolution	Real-world evidences
Focus	Run-time	Design-time	Run-time	Run-time	Both

Limitations. To validate our model, we analyzed 60 peer-reviewed studies. The model was useful for characterizing the approaches since the average number of dimensions characterizing approaches is 15 according to column A, Table 5. However, we believe that neither the model nor the comparison is unchangeable. We restricted ourselves to a limited number of high quality approaches with highest rank as it mentioned in Section IV, indicating that our comparison is not comprehensive enough, though it is reliable to cover a variety of studies. We are currently working on a more comprehensive comparison, including studies with lower ranks among the 154 ones ranked.

B. Threats to Validity

In general, the *external validity* and *construct validity* are strong for systematic reviews [22, 26, 3]. The main threats to validity of this review are *biased* by our selection of studies to be included, data extraction and data synthesis. (i) To be able to identify relevant studies and ensure that the process of selection was unbiased, a research protocol was developed. However, systematic reviews are per definition limited by search *period*, *databases* and *terminologies*. ACSE research is related to different communities including software architecture, software evolution and self-adaptive software which use different terminologies. Therefore, to cover all and avoid bias, we searched for common terms and combined them in our search string. While this approach decreases the bias, it also significantly increases the search work. (ii) To ensure correctness in data extraction, we defined a comprehensive data extraction Excel sheet [25] to obtain consistent and also relevant data. In addition, we performed quality assessment on the studies to ensure that the identified findings and implications came from credible sources. (iii) Another challenge of systematic reviews is addressing the *reliability* threats. The reliability is mitigated as far as possible by involving several researchers, and having a unified scheme, and several steps where the scheme and process were piloted and externally evaluated. If the study is replicated by another set of researchers, it is highly likely that some included papers are changed. However, it is highly unlikely that random differences based on personal judgments would change findings. It may only change the

actual count numbers to some extent. Therefore, in general we believe that the validity of the study is high, given the use of a systematic procedure and the involvement and discussion among four researchers and also external evaluations. The openness of our review by exposing our data in [25] allows other researchers to judge the trustworthiness of the results more objectively. This initiative is suggested by the evidence-based software engineering community http://www.dur.ac.uk/ebse/. It is followed by some researchers as [7].

VIII. CONCLUSIONS

The objective of this study was to consolidate existing research on architecture-centric software evolution regarding the claimed benefits and the provided evidences of evolution. The main contribution of this study is a classification framework for ACSE and a comparison of systematically selected studies through the framework to point out existing research gaps. We identified unexplored areas by synthesizing collected data, reflecting on areas of future research. The results of classification and comparison are presented as structure tables – as a means to transfer knowledge among ACSE researchers and practitioners about a collective impact of existing research.

We observed an inclination towards runtime adaptation to serve self-adaptive applications. We also observed vast interests towards probabilistic modeling and quantitative verification of system properties. Reflective environments and Models@run-time are identified as the key drivers to enable adaptations. However, evidences of rigorous validation were lacking in existing evidences in ACSE.

ACKNOWLEDGMENTS

The authors would like to thank the following persons for their feedback and thoughtful suggestions regarding the methodology, data and the final report: Jim Buckely, Jeffrey M. Barnes, Fereidoon Shams, Mehdi Fahmideh and Frank Fowley. This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

REFRENCES

[1] Mens, T., Magee, J., and Rumpe, B. 2010. Evolving software architecture descriptions of critical systems. Computer, 43(5): 42–48.

[2] Tamura, G., at al. 2012. Towards practical runtime verification and validation of self-adaptive software systems. Springer.

[3] Zhang, H., Babar, M.A. 2012. Systematic Reviews in Software Engineering: An Empirical Investigation, IST.

[4] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. 2009. Software architecture: Foundations, theory, and practice. John Wiley & Sons.

[5] Silva, L., and Balasubramaniam, D. 2012. Controlling software architecture erosion: A survey, JSS, vol. 85, no. 1, pp. 132–151.

[6] Breivold, H. P., Crnkovic, I., Larsson, M. 2011. A systematic review of software architecture evolution research, IST, 54(1), 16-40.

[7] Weyns, D., Iftikhar, M., Iglesia, D., Ahmad, T. 2012. A Survey on Formal Methods in Self-Adaptive Systems. FMSAS.

[8] Vestal, S. 1993. A Cursory Overview and Comparison of Four Architecture Description Languages. Honeywell Technology Centre.

[9] Clements, P. 1996. A survey of architecture description languages, Eighth International Workshop Software Specification and Design.

[10] Williams, B. J., Carver, J. C. 2010. Characterizing software architecture changes: A systematic review. IST 52(1): 31-51.

[11] Cruzes, D.S, and Dyba, T. 2011. Recommended Steps for Thematic Synthesis in Software Engineering, ESEM.

[12] Medvidovic, N., Dashofy, E. M., and Taylor, R. N. 2007. Moving architectural description from under the technology lamppost. IST, 49.

[13] Pahl, C., Giesecke, S., and Hasselbring, W. 2007. An ontology-based approach for modelling architectural styles, ECSA.

[14] Bencomo, N. 2009. On the Use of Software Models during Software Execution. ICSE Workshop on Modeling in Software Engineering. IEEE.

[15] Wermelinger, M., Yu, Y., Lozano, A., Capiluppi, A. 2011. Assessing architectural evolution: a case study. Empirical Software Engineering, 16(5), pp. 623–666.

[16] Zhang, H. 2010. A multi-dimensional architecture description language for forward and reverse evolution of component-based software, PhD Dissertation.

[17] Medvidovic, N., Taylor, R. N. 2000. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Trans. Software Eng. 26(1): 70-93.

[18] Buckley, J., Mens, T. Zenger, M., Rashid, A., Kniesel, G. 2005. Towards a taxonomy of software change. JSS 17(5): 309-332.

[19] Chapin, N., Hale, J. E., Kham, K. M., Ramil, J. F. and Tan, W.G. 2001. Types of software evolution and software maintenance. JSM 13(1).

[20] Zhang, P., Muccini, H., Li, B. 2010. A classification and comparison of model checking software architecture techniques. JSS 83(5), 723–744.

[21] Jeffrey M. Barnes. 2012. NASA's Advanced Multi-mission Operations System: A case study in software architecture evolution. ACM SIGSOFT Conference on the Quality of Software Architectures.

[22] Petticrew M, Roberts H. 2006. Systematic reviews in the social sciences: a practical guide. Oxford: Blackwell.

[23] ISO/IEC/IEEE 42010:2011, Systems and software engineering-Architecture description, http://www.iso-architecture.org/ieee-1471/.

[24] Calinescu, R. Ghezzi, C., Kwiatkowska, M., Mirandola R. 2012. Self-Adaptive Software Needs Quantitative Verification at Runtime.

[25] Jamshidi, P., Ghafari, M., Aakash, A., Pahl, C. A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research [Online: Auxiliary Review Material], http://www.computing.dcu.ie/~pjamshidi/SLR-ACSE.html.

[26] Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil M. 2007. Lessons from applying the systematic literature review process within the software engineering domain, JSS 80.

[27] Baresi, L., Di Nitto, E., Ghezzi, C. 2006. Toward open-world software: Issue and challenges. Comput. 39(10), 36–43.

[28] Stammel, J., Durdik, Z., Krogmann, K., Weiss, R. and Koziolek, H. 2011. Software Evolution for Industrial Automation Systems: Literature Overview, Karlsruhe Reports in Informatics.

[29] Weyns, D., Andersson, J., Malek, S., Schmerl, B. 2012. Introduction to the special issue on state of the art in engineering self-adaptive systems, JSS.

[30] Baresi, L., Ghezzi, C. 2010. The disappearing boundary between development-time and run-time. *FSE/SDP workshop,* 17-22.

[31] Aakash, A., Jamshidi, P., Pahl, C. A Classification and Comparison of Software Architecture Evolution Reuse-Knowledge [Online: Auxiliary Material], http://www.computing.dcu.ie/~pjamshidi/SLR/SLR-ERK.html.

[32] Jamshidi, P., Ghafari, M., Aakash, A., Pahl, C. 2012. A protocol for systematic literature review on Architecture-Centric Software Evolution Research, Technical Report.

[33] Ardagna, D., Ghezzi, C., Mirandola, R. 2008. Rethinking the use of models in software architecture. *Quality of Software Architectures,* 1-27.

[34] Lehman, M. 1996. Laws of software evolution revisited. *Software process technology*, 108-124.