A Framework for Acquisition and Application of Software Architecture Evolution Knowledge

Aakash Ahmad
Lero – the Irish Software Engineering Research Centre
School of Computing, Dublin City University, Ireland.
ahmad.aakash@computing.dcu.ie

Pooyan Jamshidi
Lero – the Irish Software Engineering Research Centre
School of Computing, Dublin City University, Ireland.
pooyan.jamshidi@computing.dcu.ie

Claus Pahl
Lero – the Irish Software Engineering Research Centre
School of Computing, Dublin City University, Ireland.
claus.pahl@computing.dcu.ie

ABSTRACT
Software systems continuously evolve as a consequence of frequent changes in their functional requirements and the environment surrounding them. Architecture-centric software evolution (ACSE) enables changes in software structure and behaviour while abstracting the complex implementation-specific details. However, due to recurring evolution there is a need for solutions that enable a systematic reuse of frequent changes in software architectures. In recent years, architecture change patterns and evolution styles proved successful in promoting reuse expertise to tackle architecture evolution. However, there do not exist any solutions that enable a continuous *acquisition* and *application* of architecture evolution knowledge to systematically address frequent changes in software architectures. In this paper, we propose a framework *PatEvol* that aims to unify the concepts of i) software repository mining and ii) software evolution to enable acquisition and application of architecture evolution knowledge. In the proposed *PatEvol* framework, we present knowledge acquisition (*architecture evolution mining*) to enable post-mortem analysis of evolution histories to empirically discover evolution-centric knowledge. Furthermore, we support reuse of discovered knowledge to enable knowledge application (*architecture evolution execution*) that enables evolution-off-the-shelf in software architectures. Tool support facilitates the knowledge acquisition and knowledge application processes in the *PatEvol* framework.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures

General Terms
Design, Theory.

Keywords
Software Architecture, Software Evolution, Architecture Evolution Knowledge, Evolution Patterns.

1. INTRODUCTION

Modern software continuously evolves as a consequence of frequent changes in business and technical requirements and operating environments [1, 2]. Lehman's law of continuing change [2] states that "*systems must be continually adapted or they become progressively less satisfactory*." The primary challenges while addressing continuous change [2] lie with i) acquisition and application of reusable solutions to address recurring evolution problems and ii) selection of appropriate abstractions for software change implementations [3, 4]. To address these challenges, we focus on acquisition of evolution knowledge that can be empirically discovered, shared and reused to promote evolution-off-the-shelf in software architectures.

Architectural models have proved successful in representing modules-of-code and their interconnections as high-level components and connectors to facilitate planning, modeling and executing software design and evolution at higher levels of abstraction [5, 6]. Our systematic reviews [3, 4] to analyse the state-of-research on ACSE suggest that solutions must rely on continuous acquisition of evolution-centric knowledge and expertise to guide architecture change management. In particular, architecture evolution knowledge (AEK) is defined as [3]: "*a collection and integrated representation (problem-solution map) of analytically discovered, generic and repeatable change implementation expertise that can be shared and reused as a solution to frequent (architecture) evolution problems*."

Change patterns [7, 8] and *evolution styles* [5, 6] promote the 'build-once, use-often' philosophy to address a continuous evolution in software architectures. However, a systematic analysis of existing research [3, 4] highlights the need for solutions that enable integration of evolution-centric knowledge acquisition [9] that guides knowledge application [10] to evolve software architectures. We propose to unify the concepts of a) software repository mining [11, 12] (for knowledge acquisition) and b) software evolution [1, 2] (for knowledge application) to address the problems of frequent changes in ACSE - presented in Figure 1. We propose a framework *PatEvol* that provides an integration of knowledge acquisition and knowledge application processes to facilitate reuse of evolution knowledge. By process integration, we mean that the architecture evolution mining process enables a continuous acquisition of evolution-centric knowledge by analysing architecture evolution histories, and then discovered knowledge can be applied to support architecture evolution execution.

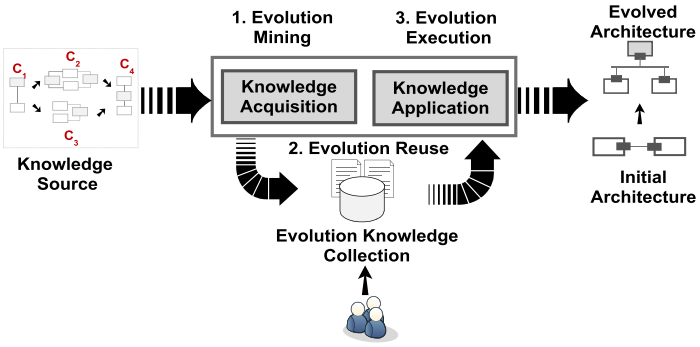

Figure 1. Overview of Architecture Evolution Knowledge Acquisition and Application Processes

The *PatEvol* framework comprises a set of processes and activities to enable acquisition and application of evolution knowledge. The outcome of this paper is a novel framework that aims to support:

A. Acquisition of Architecture Evolution Knowledge – also referred to as *architecture evolution mining* and detailed in Section 3. It enables the post-mortem analysis of architecture evolution histories to discover evolution-centric knowledge. In Figure 1, an architecture evolution history is represented as a source of knowledge that consists of traces of architecture-centric changes maintained during evolution of software architectures. *Knowledge Source* represents a transparent and centrally

manageable repository [11, 12] as a recorded collection of architecture change representations [9, 13]. It provides a foundation and fine-grained change representations for experimental analysis of real data concerning architecture evolution. The establishment and existence of a knowledge base is a fundamental requirement in capturing architectural changes as an experimental foundation for knowledge discovery.

B. Application of Architecture Evolution Knowledge – is also referred to as *architecture evolution execution* and detailed in Section 3. It enables the utilisation of knowledge discovered during the evolution mining process to enable reuse of generic expertise to enable architecture evolution. In Figure 1, evolution execution is characterised by changes in source architecture – application of addition, removal and modification operations – to enable its evolution [5]. However, evolution is not just the addition or removal of architecture elements; among other tasks, it also requires evolution plans and tradeoff analyses [5], preserving the structural integrity of architecture elements and exploiting architecture change composition [6]. An *Evolution Knowledge Collection* represents a knowledge base as an active repository that contains a collection of empirically discovered evolution knowledge. Knowledge collection is therefore vital in order to absorb the evolution-centric knowledge that could be shared and reused across multiple evolution problems to guide architecture evolution. In Figure 1, traces of architecture evolution are captured in the knowledge source enabling knowledge discovery and application as a continuous process.

The rest of this paper is organised as follows. In Section 2, we provide an overview of some existing reference frameworks for architecture-driven modernisation and evolution to justify the novelty and contribution of the proposed *PatEvol* framework. We discuss the processes, activities and repositories for acquisition and application of architecture evolution knowledge in Section 3 and present our conclusions in Section 4.

2. FRAMEWORKS FOR ARCHITECTURE MODERNISATION AND EVOLUTION

In software engineering and software evolution literature, the terms modernisation and evolution are virtually synonymous and often used interchangeably – referring to architecture-based change implementation [1, 13]. In this section, we explain some existing frameworks that are used as reference models to guide the architecture evolution process. We specifically discuss the *Architecture Driven Modernisation* (ADM) framework [14] and the *SOA Migration Horseshoe* Model [15] for architectural migration and evolution. A brief explanation of these frameworks is vital to highlight the contributions of our proposed *Pattern-Driven Architecture Evolution* (*PatEvol*) framework. Both the ADM framework and the SOA migration Horseshoe model are conceptual extensions of the famous Horseshoe model for architecture-based reverse engineering [16] proposed by the Software Engineering Institute (SEI).

The selection of the two above-mentioned frameworks also helps with a high-level assessment of the *PatEvol* framework and its underlying activities to support architecture evolution. For comparative analysis, we selected the ADM and SOA Migration Horseshoe models because both of them represent research with appropriate citations, availability of documentation and details about a structured set of activities for architecture migration and evolution. The concepts and methods used in these reference models can be reused or possibly extended to develop the processes and activities in the *PatEvol* framework. More specifically, method engineering [17] enables us to reuse the existing concepts from existing methods (frameworks, models or solutions) to develop new methods by reusing existing methodologies with reduced effort and time. In the following, we highlight the role of ADM [14] and SOA migration Horseshoe [15] in architecture evolution, which provides us a foundation to discuss the technical details of the *PatEvol* framework in Section 3.

2.1 Architecture-Driven Modernisation Framework

The ADM framework [14] is a conceptual extension of the Horseshoe model for architecture-based reverse engineering [16] proposed by SEI, illustrated in Figure 2. The ADM model transforms the existing architecture towards the target architecture by maintaining a layered view of three different levels of architectural abstractions. The three architectural layers in ADM are called *Technical Architecture* layer, *Application and Data Architecture* layer and *Business Architecture* layer. The existing architecture is represented on the left while the evolved or the target architecture across all three layers is represented on the right. The horizontal arrow from existing to target architecture represents transformation-driven architectural evolution. Transformations in the ADM framework involve an incremental evolution from existing to target architecture at any layer. For example, evolution at the technical architectural level involves source code transformation (e.g. procedural to object oriented transformation) of legacy code. In summary, transformation at any architectural layer relies on three elements:

1. *Knowledge discovery* of the legacy system,
2. *Definition* of target architecture, and
3. *Transformation* steps for source to target evolution.

The ADM framework provides a comprehensive reference model for architectural transformation and modernisation at three different layers of abstraction. The evolution can be at any architectural abstraction level: from the code level transformation (e.g. source code refactoring for migration) to more abstract and conceptual levels (e.g. software design, evolution, and business-rule transformation etc.). An inherent limitation with such a comprehensive framework lies with the diverse scope of evolution activities (source code refactoring, software architecture evolution, business model transformations). In addition, the framework does not consider the frequency of architecture evolution. Due to the complexities involved with different architectural abstractions, it is difficult to reuse transformations across different layers of the ADM framework to tackle frequent architecture evolution.

2.2 SOA Migration Horseshoe Model

The SOA Migration Horseshoe model [15] is also a specialised derivation of a general Horseshoe model for architectural reverse engineering [16]. The model integrates software reengineering and business process modeling and aims to:

1. Exploit reverse engineering techniques to extract the *Legacy Enterprise Model* from the *Legacy Source Code*.

2. Apply enterprise modeling techniques to create a *Consolidated Enterprise Model* and to identify *Services* using forward engineering techniques.

3. Map the *Legacy Code* to *Services* via wrapping or transformation of *Components*.

In the context of software architecture models and their evolution, the SOA migration horseshoe model is of a less technical nature; it focuses more on business engineering aspects of enterprise software and its architecture. The underlying question it tries to ask is *how can a sub-functionality be identified as a potential service, or how can business process models be derived from a legacy system*.

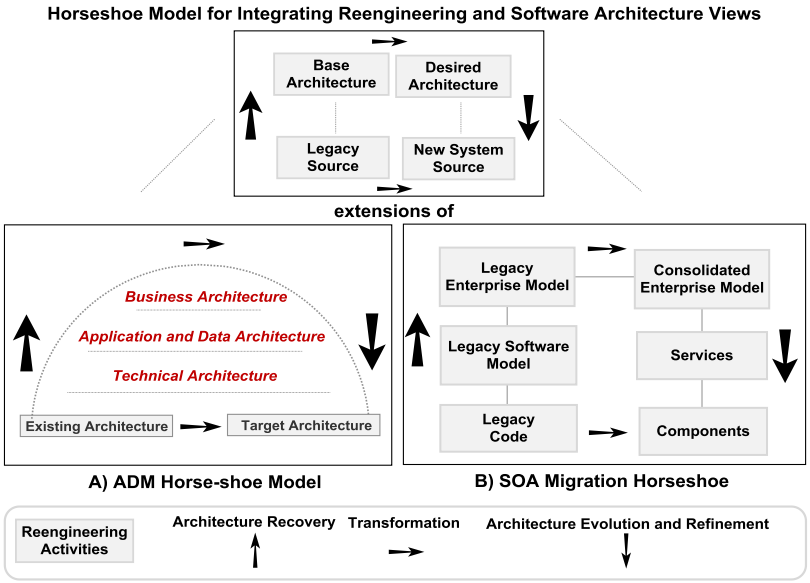

Figure 2. Overview of the ADM Framework and the SOA Migration Horseshoe Model

In contrast to the reengineering Horseshoe [16], ADM [14] and SOA migration models [15], the proposed *PatEvol* framework (detailed in Section 3) is limited to addressing evolution only at the architectural level of abstraction. *PatEvol* only considers reuse of evolution-centric knowledge that is not addressed in any of the existing frameworks. More specifically, in the context of architecture change analysis and management, the *PatEvol* framework lies at the intersection of two distinct research areas: i) *software repository mining* [11, 12] to discover evolution knowledge from architecture evolution histories [13] and ii) *software evolution* [1, 2] that relies on the discovered knowledge to support architectural evolution [10]. In conceptual terms, the *PatEvol* framework adapts the basic ideas from the SOA migration horseshoe [15]. However, in contrast to legacy migration towards service-based software, it is focused on systematically accommodating the new requirements in existing architectures that support a reuse-centered approach to achieve ACSE.

3. PatEvol – A FRAMEWORK FOR ACQUISITION AND APPLICATION OF ARCHITECTURE EVOLUTION KNOWLEDGE

In the *PatEvol* framework, we propose acquisition of architecture evolution knowledge as a complementary and integrated process to knowledge application, as illustrated in Figure 3. In the remainder of this section, we discuss the *processes*, *activities* and *repositories* as the building blocks of the *PatEvol* framework. We primarily focus on:

– *Knowledge Acquisition* is achieved with *architecture evolution mining* that represents a sub-domain of software repository mining and enables an (automated) extraction of hidden and predictive information from large data sets regarded as software evolution histories [11]. Evolution mining is particularly beneficial for establishing and utilising an experimental foundation for the 'post-mortem' analysis of evolution histories to discover reusable operations and patterns of evolution.

– *Knowledge Application* is achieved with *architecture evolution execution*, which refers to a systematic mapping among the problem-solution views and the application of the discovered solutions to recurring problems of architecture evolution [5, 6].

3.1 Processes, Activities and Repositories in the PatEvol Framework

In this section, we provide details about the main building blocks of the *PatEvol* framework. Each conceptual element is presented along with its role in the framework as summarised in Table 1 and illustrated in Figure 3. We propose *PatEvol* as a conceptual framework that outlines a set of processes and activities to enable discovering and reusing evolution knowledge. The processes in the framework define *what* needs to be done and the activities in a process demonstrate *how* it is done [26]. A top-down view of the framework is presented in Figure 3 with a summary of processes and activities in Table 1. In the following, we discuss the underlying concepts in terms of framework *processes*, *activities* and *collections* along with the *transitional* steps among the activities and processes.

Table 1. Processes, Activities and Repositories in the *PatEvol* Framework

Processes	Process Activities	Repositories
Knowledge Acquisition	Classification of Architecture Changes	Architecture Change Logs
	Discovery of Architecture Evolution Patterns	
	Specification of Evolution Patterns	
Knowledge Execution	Specification of Architecture Evolution	Evolution Patterns Catalogue
	Selection of Architecture Evolution Patterns	
	Pattern-based Reuse of Architecture Evolution	

– *Processes in the Framework*: The processes (indicated as a white square) represent two distinct parts of the framework as a) knowledge acquisition (enabled through evolution mining) and b) knowledge application (enabled through evolution execution) as in Table 1.

– *Activities inside Processes*: Each process comprises a set of underlying activities (indicated by a blue rectangle) that highlight the distinction between knowledge discovery and its application in evolution. Both of the knowledge acquisition and knowledge application processes are comprised of three activities as in Table 1.

– *Role of Repositories in the Framework*: In addition to the core processes and activities, the role of repositories or knowledge collections is vital as the source and sink of evolution knowledge. More specifically, the knowledge source or architecture change logs [9, 13] represent a central repository that contains fine-grained instances of architecture change and provides a foundation for evolution mining. We propose a catalogue of architecture evolution patterns that promotes an empirically discovered collection of patterns as reusable solutions to recurring problems of software architecture evolution.

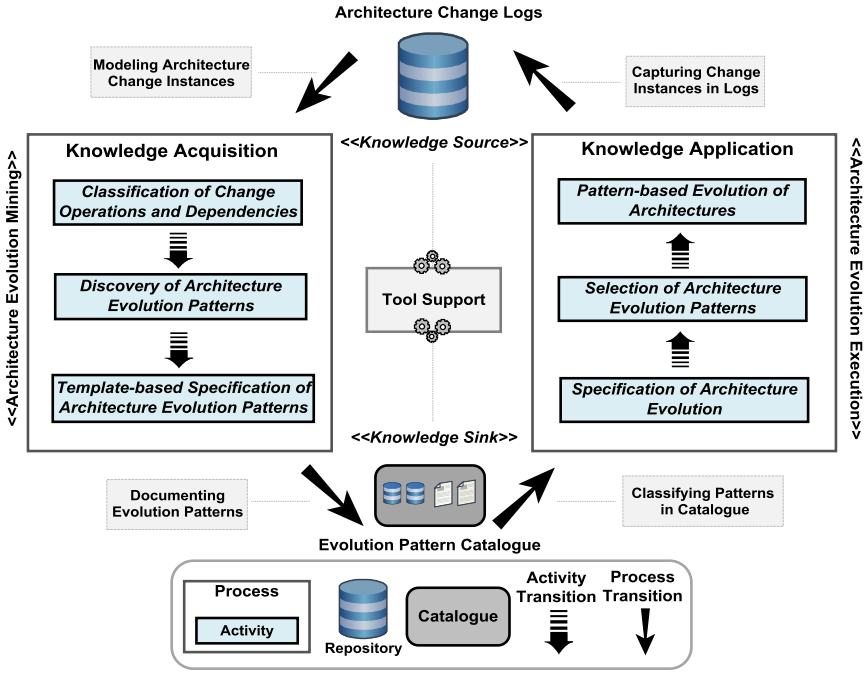

Figure 3. The PatEvol Framework – An Integrated View of Knowledge Acquisition and Knowledge Application Processes

Finally, the transitions among the processes and activities are represented as the activity and process transitions arrows that reflect stepwise and incremental approaches to extracting, representing and utilising architecture evolution knowledge. In the following, we summarise the overall objective of the *PatEvol* framework [25] that aims to consolidate the corresponding activities that complement discovering and reusing evolution-centric knowledge as presented in Figure 3.

3.2 Framework Processes and Activities

The framework consists of two main processes. A set of activities defines the atomic production steps of a process that aim to achieve the objectives of the process in an incremental and stepwise manner [26]. In addition, a brief discussion of processes and activities allows us to highlight the contributions of the *PatEvol* framework.

Process I – Acquisition of Evolution Knowledge

The role of knowledge acquisition is fundamental in enabling a systematic investigation into the history of sequential architecture evolution for analysing recurring changes. Acquisition of evolution knowledge is achieved with architecture evolution mining. Our objective for evolution mining is identical to that of software evolution analysis [12, 11], which exploits the history of a software system to analyse its present state and to predict its future. In the context of software repository mining, architecture evolution mining is aimed at employing a set of (automated) techniques for extraction of architecture change instances from change logs [9, 13]. Therefore, we exploit architecture change logs that provide fine-grained details about architecture change instances. The change instances may vary from a simple change like adding a port into a component to a complex change like integrating, replacing or decomposing components in an existing architecture. In a collaborative environment for architectural development and evolution, the change log represents a knowledge source to facilitate post-mortem analysis for architectural evolution [13].

Modeling Architecture Change Instances from Logs

In order to systematically investigate change logs, we need to formalise individual change instances captured in the log. The need for a formal and structured representation is driven by the fact that raw representation of log data is complex, and therefore its analysis is time consuming and error prone. We exploit a graph-based notation to formalise change instances in the log as graphs [11] with nodes and edges capturing change operations on architecture elements. A graph-based representation of the log data is beneficial for a formal (semi-) automated and efficient analysis of fine granular change instances in the logs. In addition, when modeling architecture changes as graphs, a significant benefit lies in utilising sub-graph mining [18] techniques. By applying graph mining to architectural changes, we can discover recurring sub-graphs (sequences of change operations) that represent frequent evolution patterns in a formal and automated way. The goal of this activity is to formalise the change log data that is represented as an architecture change graph.

In the following, we discuss the activities of the framework that are focused on log-based *taxonomical classification of architecture change operations and operational dependencies*. The ultimate outcome of the evolution mining is *discovery* of architecture evolution patterns and their specifications, which provide the foundations to develop an *evolution pattern catalogue* to promote reuse of recurring architectural evolution tasks.

Activity I –Taxonomical Classification of Architecture Change and Operational Dependencies

Once log data is formalised as a graph [9], a more intuitive approach to gain a systematic insight into architectural changes is to analyse how changes are represented over a period. The graph-based formalism provides us with an option to exploit graph-matching – comparing change instances – to analyse the operational composition and characterisation of changes [18]. Such an analysis requires details about the composition of architecture changes and the possible operational representations of change instances. This is beneficial to recover and taxonomically classify changes based on their complexity as either *atomic* or *composite* [13]. The dependencies among change operations are classified as *commutative* and *dependent* change operations [13]. Change dependency analysis helps us to analyse *the extent to which architectural change operations are dependent or independent of each other* (*whether architecture change operations could be parallelised*). The outcome of this activity is a taxonomical classification of change instances as either *atomic* or *composite* change operations. In addition, a fine-grained change operational classification

is vital to distinguish between *commutative* and *dependent* changes in architecture evolution.

Activity II – Discovery of Architecture Evolution Patterns

The outcome of activity I is a taxonomical classification of architecture change operationalisations that provides a foundation to discover the frequency of change operation sequences in the log. The frequency of change determines whether a certain type of change occurs repeatedly. This motivates us to exploit change sequence abstraction to determine frequently occurring changes that represent potential *evolution patterns* discovered from change logs [9]. An evolution pattern represents a generic and potentially reusable operationalisation that could be i) identified as a recurrent solution, could be ii) specified once and iii) instantiated multiple times to support potential reuse in architecture evolution [10]. The outcome of the pattern discovery activity is a collection of discovered patterns from logs that allow us to develop a catalogue of architecture evolution patterns.

Activity III – Template-based Specification of Evolution Patterns

After pattern discovery, we need to provide a consistent and once-off specification of architecture evolution patterns in the catalogue. Pattern specification allows us to share and reuse the discovered patterns. We follow the guidelines for pattern documentation in [19] for a template-based specification of architecture evolution patterns. A pattern template provides a structured document to capture the intent and consequences of pattern application. A template-based pattern specification provides a collection of change patterns that support reusable solutions to recurring evolution problems. We believe that by exploiting the patterns in change catalogues, individual patterns can be formalised and interconnected to support reusable, off-the-shelf evolution. Evolution knowledge in the catalogue is expressed as a collection of evolution patterns. It is vital to mention that, patterns as a generic and solution-specific knowledge to resolve recurring evolution problems could not be invented. Instead, patterns along with their possible variants must be discovered by analysing the problem space and the solution context [13]. We summarise the outcome architecture evolution mining process as:

– Enabling 'post-mortem' analysis of architecture evolution histories to discover patterns that could be shared and reused to guide architecture change management.

– Template-based specification [19] of discovered patterns enable problem-solution mapping to reuse generic operationalisation. The role of the pattern catalogue is central in promoting patterns to achieve reuse and consistency in architecture evolution.

Process II – Application of Evolution Knowledge

In the context of software evolution [1, 2, 3], architecture evolution execution refers to a systematic implementation of architectural changes as an addition, removal, and modification of elements to modify an existing architecture [5, 6]. Because of frequent business and technical change cycles, software systems and ultimately their architectures tend to require continuous maintenance and evolution. This motivates the need to unify the concepts of data mining or more specifically software repository mining and software evolution in a way that evolution mining provides discovered knowledge used to complement and guide evolution execution. Such an integrated approach is missing in the existing solutions [5, 7, 14, 15] and enabling it relieves an architect of routine evolution tasks by fostering their reuse to support a systematic change execution whenever needs for architectural evolution arise [2]. In the context of evolution execution in Figure 3, evolution patterns provide a knowledge base for pattern-driven architecture evolution. During evolution, change instances are captured for an incremental update of evolution history to establish the loop for knowledge acquisition and knowledge application [3]. In the following, we discuss the activities of the knowledge application process that represent: i) a declarative specification of architecture evolution to select ii) a list of appropriate patterns from the catalogue and to enable iii) pattern-driven reuse in architecture evolution.

Activity I – Specification of Architecture Evolution

Evolution specification allows representing the changes to a source architecture that leads to its evolution [10]. In this context, a declarative specification enables an architect to represent the syntactical context of architectural evolution that contains the i) source architecture ii) any constraints on the architecture model and iii) specific architecture elements that need to be added, removed or modified to achieve architecture evolution. In addition to a syntactical context, evolution specification allows us to represent the intent and scope of individual changes explicitly in the source architecture model. During evolution specification an architect may want to specify architectural constraints to preserve the specific architectural elements from consequences of change before and after evolution. In order to enable evolution, a specification of architectural changes is the first step to represent a transition of source architecture towards an evolved architecture.

Activity II - Selection of Architecture Change Patterns

Once architectural changes are specified, the pattern catalogue provides a collection of patterns as problem-solution mapping based on a given evolution context. However, pattern selection is a complex problem [20] and in order to query the catalogue the user must know the internal structure of the pattern catalogue as well as a detailed knowledge about existing patterns in the catalogue collection. We adopt the design space analysis [21] for a systematic pattern selection from the catalogue. Design space analysis is a methodology to address design-related problems in Human Computer Interaction (HCI). Following design-space analysis, change specification enables querying the catalogue using the Question-Option-Criteria (QOC) methodology [21] to retrieve the appropriate pattern(s) that provides the potential reuse of architectural evolution. More specifically, in QOC *Question* refers to declarative specification of architectural changes, *Option* represents the available patterns in a given evolution scenario, and *Criteria* represent the consequences and impacts of the given pattern.

Activity III – Pattern-based Evolution of Architectures

The retrieved pattern(s) could be applied to abstract the operational execution thus supporting reuse in architectural change execution. In addition to pattern retrieval, pattern application or instantiation involves labeling of generic elements in the specification with labels of concrete architecture elements presented in change specification. With a pattern-driven architecture evolution approach, we claim that *if an architectural evolution problem can be specified declaratively, then its solution is executed in an automated way by instantiating change operationalisations that exists in the pattern catalogue*. The ultimate outcome of the change execution process is:

– A declarative specification of change requests that enables selection of appropriate pattern sequences to derive reusable evolution strategies based on given evolution scenarios.

– The pattern catalogue provides a method of systematic reuse based on an incremental application of patterns from the collection.

3.3 Collection Types in the PatEvol Framework

We discuss the processes and their underlying activities that enable integration among architecture evolution mining and architecture evolution execution processes. In this integration the role of repositories in the framework as an architecture evolution history and evolution patterns collection could not be overlooked. In the *PatEvol* framework, the role of these repositories is central as the knowledge source in terms of extracting change instances in evolution mining and fostering reusable operationalisations during evolution execution.

Repository I - Change Log as a Source of Architecture-Centric Evolution Knowledge

In order to ensure an incremental discovery of evolution knowledge, it is required to capture and maintain the traces of evolution by means of a transparent and centrally manageable collection of change instances [13, 9]. In a conventional context, change related data is extracted from versioning systems [11], as their repositories contain the artifacts that designers and developers produce and modify. The granularity of information contained in versioning systems is not complete enough to perform higher quality evolution research. Since the past evolution of a software system is not a primary concern for most developers, it is not an important requirement when designing versioning systems [11, 12]. On the contrary, the details of information stored in a change log [13, 9] can be exploited to capture fine-grained instances of change operations on individual architecture elements. In order to provide an experimental foundation for evolution analysis, the architecture change log provides a source of evolution knowledge that can be shared and reused.

Collection II – Catalogue as a Collection of Architecture Evolution Patterns

An *evolution pattern* [9] is a recurring solution to common problems in a given evolution context, resolving a set of consequences and forces. The potential beyond individual patterns is realised as a collection of change patterns that represent a generic and potentially reusable solution to a set of evolution problems [10]. In this context, an *evolution pattern catalogue* is collection of patterns to solve the prevalent problems in the architecture evolution context. As an integrated solution, in Figure 3, we propose *evolution mining* to empirically discover explicit evolution knowledge as patterns that can be maintained in the catalogue for reuse whenever needs for architecture evolution arise. As a contrary to pattern invention in [7, 8], we investigate architecture change logs [13] to empirically discover a classified composition of evolution patterns and possible variants.

4. CONCLUSIONS AND OUTLOOK

In this paper, we presented a framework for a continuous acquisition of architecture evolution knowledge and its application to support architectural maintenance and evolution. In order to realise the research potential, we proposed *PatEvol* as a framework that focuses on enabling pattern-driven reuse in architecture-centric software evolution. The framework aims to unify the concepts of architecture evolution mining as a complementary and integrated phase to architecture evolution execution. We summarise the ultimate benefits of using the *PatEvol* as:

- Exploiting architecture change logs (histories of sequential changes) to continuously identify *architecture evolution patterns* that provide *generic solutions to recurring architecture evolution problems*.
- Support for pattern specification and instantiation through a *pattern catalogue* that consists of a *continuously validated and updated collection of patterns* as reusable solutions to architecture evolution problems.
- An *evolution application framework* to enable pattern-based reuse during change execution to support the notion of *off-the-shelf evolution* in software architectures.
- At the core of the *PatEvol* framework is a discovery of evolution patterns to continuously feed the catalogue.

5. REFERENCES

[1] T. Mens and S. Demeyer. Software Evolution. Springer, 1st Edition, 2008.

[2] M. Lehman. Laws of Software Evolution Revisited. In Software Process Technology, LNCS 1996.

[3] A. Ahmad, P. Jamshidi and C. Pahl. Classification and Comparison of Evolution Reuse Knowledge in Software Architectures - A Systematic Review. In Technical Report, School of Computing, Dublin City University, pages 1–30, 2013. Available from: http://www.computing.dcu.ie/~pjamshidi/SLR/SLR-ERK.html.

[4] P. Jamshidi, M. Ghaffari, A. Ahmad, C. Pahl. A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research. In 17th European Conference on Software Maintenance and Reengineering, 2013.

[5] J. M. Barnes, D. Garlan and B. Schmerl. Evolution Styles: Foundations and Models for Software Architecture Evolution. In Journal of Software and Systems Modeling, 2012.

[6] O. L. Goaer. D. Tamzalit, M. Oussalah, A. D. Seriai. Evolution Shelf: Reusing Evolution Expertise within Component-Based Software Architectures. In IEEE International Computer Software and Applications Conference, 2008.

[7] K. Yskout, R. Scandariato, W. Joosen. Change Patterns: Co-evolving Requirements and Architecture. In Journal of Software and Systems Modeling, 2008.

[8] I. Côté, M. Heisel, I. Wentzlaff. Pattern-Based Evolution of Software Architectures. In European Conference on Software Architecture, 2007.

[9] A. Ahmad. P. Jamshidi, C. Pahl. Graph-based Pattern Identification from Architecture Change Logs. In 10th International Workshop on System/Software Architectures, 2012.

[10] A. Ahmad, P. Jamshidi, and C. Pahl. Pattern-driven Reuse in Architecture-centric Evolution for Service Software. In 7th International Conference on Software Paradigm Trends, 2012.

[11] H. Kagdi, M. Collard and J. Maletic. A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. Journal of Software Maintenance and Evolution: Research and Practice, 2007.

[12] T. Zimmermann, A. Zeller, P. Weissgerber and S. Diehl. Mining Version Histories to Guide Software Changes. In IEEE Transactions on Software Engineering, 2005.

[13] A. Ahmad, P. Jamshidi, M. Arshad, C. Pahl. Graph-based Implicit Knowledge Discovery from Architecture Change Logs. In 7th Workshop on SHaring and Reusing Architectural Knowledge, 2012.

[14] W. M. Ulrich, P. Newcomb. *Information Systems Transformation: Architecture-Driven Modernization Case Studies*, Morgan Kaufmann Publishers Inc, 2010.

[15] A. Winter and J. Ziemann, "Model-based Migration to Service-Oriented Architectures. In International Workshop on SOA Maintenance and Evolution, 2007.

[16] R. Kazman, S.Woods, J. Carriere. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In Working Conference on Reverse Engineering, 1998.

[17] S. Brinkkemper. Method Engineering: Engineering of Information Systems Development Methods and Tools. In *Information and Software Technology*, 1996.

[18] C. Jiang, F. Coenen, and M. Zito. A Survey of Frequent Subgraph Mining Algorithms." *Knowledge Engineering Review*, 2012.

[19] N. B. Harrison, P. Avgeriou, U. Zdun. Using Patterns to Capture Architectural Decisions. IEEE Software 24(4): 38-4, 2007.