Towards a Catalog of Refactoring Solutions for Enterprise Architecture Smells

Lukas Liss[1], Henrik Kämmerling[1], Peter Alexander[1] and Horst Lichter[1]

[1]RWTH Aachen University, Research Group Software Construction, Ahornstrasse 55, Aachen, Germany

Abstract

The model of enterprise architecture (EA) is often a primary means of steering the business-IT development. It helps to ensure EA qualities in many ways, including identification of weaknesses in an EA or the signs thereof. Such weaknesses have been addressed through the study of EA smell, which focuses on common bad habits in EA practices. Although EA smell problems have been described in various contexts, current solutions lack the basis of evidence and practical details that are necessary for their adoption. Therefore, this study seeks to gain new insights into the solution domain of EA smells by exploring current knowledge about refactoring solutions. We present our findings in a catalog of EA refactoring solutions intended to serve as food for thought for future research directions.

Keywords

enterprise architecture, enterprise architecture smell, refactoring solution

1. Introduction

The model of enterprise architecture (EA) greatly supports sustainable management of complex IT landscapes. It provides insights into the current implementations and future orientations of the EA developed, thereby providing a reasoning basis for important decisions. Nevertheless, the maintenance of the EA model is often pushed down the priority list due to, e.g., lack of resources or supporting methods. Flaws and deficiencies in the EA may remain ignored and create barriers to EA evolution known as EA debt [1]. To prevent such tendencies, practical methods for supporting the continuous evaluation and improvement of EA models are needed.

Using EA models to guide the improvement of EA qualities has been proposed by some studies. Salentin and Hacks coined the concept of EA smell to address common bad habits in EA practices and derived EA smells [2] by transferring known code smells into the context of EA. Lehmann et al. made a similar attempt to derive process modelling anti-patterns for EA analyses [3] by transferring known workflow anti-patterns into the context of EA. Both of these studies suggest, among others, some solutions to refactor the problems they identify. However, existing descriptions of these solutions lack the basis of evidence and practical details that are necessary for their application. This study therefore focuses on exploring current knowledge about refactoring solutions to find further evidence about EA refactoring solutions and new ways of approaching them.

Considering that current EA smells were derived from code smells, we argue that exploring known code refactoring solutions can result in meaningful insights into the refactoring solutions for EA smells. Therefore, based on the code smells used in EA smell studies, we first searched for relevant code refactoring solutions in major code smell and refactoring catalogs. The code refactoring solutions collected were then used to answer the following research questions (RQ):

RQ 1 What EA refactoring solutions can be derived from the existing code refactoring solutions?

RQ 1.1 How to derive insights about EA refactoring from analyzing code refactoring solutions?

RQ 1.2 What attributes can describe an EA refactoring solution?

RQ 2 How can EA practitioners and researchers benefit from knowing about EA refactoring solutions?

The remainder of this paper is structured as follows: Section 2 gives an overview of studies related to refactoring solutions. Section 3 describes our methodology for deriving and documenting refactoring solutions for EA smells. Section 4 presents the resulting EA refactoring solutions mapped to the corresponding EA smells. Section 5 demonstrates some small practical examples of using EA refactoring solution for mitigating EA smells. Section 6 discusses our finding, its implications, and the threats to its validity. Section 7 motivates future research directions and concludes this paper.

QuASoQ 2021: 9th International Workshop on Quantitative Approaches to Software Quality, December 06, 2021, Taipei, Taiwan
lukas.liss@rwth-aachen.de (L. Liss);
henrik.kaemmerling@rwth-aachen.de (H. Kämmerling);
alexander@swc.rwth-aachen.de (P. Alexander);
lichter@swc.rwth-aachen.de (H. Lichter)
0000-0001-6534-278X (P. Alexander); 0000-0002-3440-1238 (H. Lichter)

2. Related Work

The concept of refactoring has been developed to support software design and evolution, specifically to rectify design flaws and improve software quality through restructuring plans while preserving the observable behavior. The term 'refactoring' can be used in different ways, namely 'refactoring' (noun) to mention the restructuring applied to a software [4], 'to refactor' (verb) to mention the activity of restructuring a software [4], or 'refactoring solution' to mention a possible technique for restructuring a software (e.g. [5]). For the sake of clarity, these ways of usage are applied in this paper.

After being extensively used in programming domains, such as the procedural programming (e.g. [6]), object-oriented programming (e.g. [5]), and functional programming (e.g. [7]), the concept of refactoring has been explored in a wider spectrum of software engineering. In the domain of software modelling, the concept of model refactoring was introduced to enable restructuring plans on the high-level description of software [8]. The aim of model refactoring can be twofold: to allow for an earlier (i.e. at the design stage) and easier (i.e. reduced complexity) refactoring of software; or to improve semantic and syntactic quality measures of the model. Whichever aim is set, model refactoring should preserve both the behavior of the modelled software and the semantic of the model.

Studies of model refactoring have varied in their focus, whether it be on a specific modelling language or particular architectural aspect. Modelling languages such as the object constraint language (OCL), unified modelling language (UML), and web ontology language have been studied and supported with designated refactoring solutions [9] [10] [11]. Architectural aspects such as process aspect, data aspect, and style aspect have also been analyzed in this context to support a comprehensive architecture restructuring [12] [13] [14]. Furthermore, the so-called large refactorings have been proposed to deal with refactoring in complex projects, specifically when changing significant parts of a system, which often takes longer than a day [15]. Still, there are growing possibilities to explore new refactoring solutions for new design anomalies, especially with the growing interest in bad smell research—which focuses on detecting concrete indication of the need for refactoring [4].

The concept of bad smell has been adopted in the domain of EA and referred to as EA smell, which represents negative examples and bad habits that, when ignored, may harm the performance of EA activities or even the organization as a whole [2]. The concept was proposed together with a catalog of 45 EA smells, which describes (among others) the problem contexts, applicable detection methods, and possible mitigation solutions. Based on the scope of occurrence, an EA smell can be of the business, application, and/or technology architecture. This scope was recently expanded by the exploration into EA process anti-pattern, which resulted in 18 EA smells for process-related issues [3]. As the interest in EA smell research is growing, more EA smells will continue to be identified through new explorations into other aspects of EA, such as management.

3. Methodology

This study uses the design science research (DSR) methodology according to the guidelines proposed by Peffers et al. [21] and Hevner et al. [22]. A DSR aims to devise an artifact that addresses a "heretofore unsolved and important business problem" by drawing on the existing knowledge, and the resulting artifact must be rigorously evaluated in terms of its "utility, quality, and efficacy" and effectively communicated to relevant audiences. With respect to the DSR guidelines above, this study follows the six main steps of DSR as listed below.

1. **Problem identification and motivation.** Define the specific research problem and justify the value of the solution proposed
2. **Solution objective definition** Infer the objectives of the solution proposed from the problem definition and knowledge of what is possible and feasible
3. **Design and development.** Create the artifact
4. **Demonstration.** Demonstrate the use of the artifact to solve one or more instances of the problem
5. **Evaluation.** Observe and measure how well the artifact supports a solution to the problem
6. **Communication.** Communicate the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences

The following subsections describe the process and methods employed in this study with respect to the steps above.

3.1. Problem identification and motivation

Despite current results in EA smell research (as presented in section 2), the identification of EA smells is never an end in itself. The identified EA smells should, e.g., inform the decisions for imposing suitable EA improvement measures with regards to the current circumstances and interests. To reason such decisions, enterprise architects need to first understand the applicability of each solution alternative through evidence and practical details,

UML → ArchiMate	Source	UML → ArchiMate	Source
Activity → Interaction, Function, Process	[16, 17, 18]	**Interface** → Interface, Service	[19, 16, 17, 20, 18]
Actor → Actor, Role	[19, 16, 17, 20]	**Method** → Application/Infrastructure Function, Infrastructure Service	[16]
Artifact → Artifact, Contract, Deliverable, Gap, Product, Representation	[19, 16, 17, 20]	**Note/Comment** → Meaning, Representation	[18]
Association → Association, Aggregation, Composition	[17, 20, 18]	**Node** → Node, Communication Path, Device, Infrastructure Interface, Location, Network, System Software	[19, 16, 17, 20]
Class → Actor, Collaboration, Component, Data Object, Meaning, Motivational Concepts, Object, Plateau, Representation, Role, Stakeholder, Value	[19, 16, 20, 18]	**Object** → Object, Contract, Data Object, Meaning, Product, Representation, Value	[16]
Collaboration → Collaboration, Function, Interface, Location	[19, 16, 17, 20, 18]	**Opaque Behavior** → Application Interaction, Business Event, Business Interaction, Business Process, Junction, Work Package	[20]
Component → Component, Grouping	[19, 16, 17, 20]		
Communication Path → Communication Path, Network	[17, 20, 18]	**Swimlane** → Actor, Business Service, Role	[16]
Dependency → Assignment, Derived, Influence, Serving/Used by	[17, 20]	**Usage** → Access, Serving/Used By	[20]
Device, Event, Execution Environment → Device, Event, System Software	[19, 16, 17, 20]	**Use Case** → Business Function, Business Interaction, Requirement, Service	[19, 16, 17, 20]
Generalization → Specialization	[19, 20, 18]	**Realization, Composition, Aggregation** → Realization, Composition, Aggregation	[19, 17, 20, 18]
Information Flow → Flow, Triggering	[20]		
Interaction → Interaction, Application Service, Event, Function, Process	[16, 18]		

Table 1
A mapping from UML to ArchiMate

which are still lacking in the hitherto solution suggestions for EA smell mitigation. Furthermore, current knowledge about the refactoring of architectural smells focuses on rather technical aspects (i.e. software architecture), which are hardly applicable for evaluating and evolving an EA. Therefore, we argue that EA research should pursue the identification of possible EA refactoring solutions to guide enterprise architects in finding possible and feasible solutions for the EA smell problems at hand.

3.2. Solution objective definition

Considering the problem described above, this study aims to devise the concept of EA refactoring solution by drawing on the existing knowledge about refactoring. To guide the solution design and development, this study sets the following solution objectives:

1. Support enterprise architects in finding appropriate solutions to a certain EA smell by identifying possible EA refactoring solutions and providing clear descriptions about their context and implementation
2. Support enterprise architects in communicating relevant EA refactoring solutions in a consistent manner by identifying relevant attributes and providing a standard documentation template
3. Allow the EA community to obtain an up-to-date overview of results in EA refactoring research and contribute to the its development

3.3. Design and development

Since the first EA smells were derived from code smells [2], we assume that refactoring solutions for code smells may hold some clue to the refactoring solutions for EA smells. This assumption leads to the design of our methodology, which includes three main steps: Concept analysis, concept mapping, and concept transformation.

Concept Analysis. Our first step is to collect relevant code refactoring solutions for the code smells used by existing studies on EA smells [2]. Our analysis covered some existing code smells or anti-patterns catalogs (i.e., [4], [23], and [24]). While we are aware that "the presence of code smells does not imply the presence of architectural smells and vice versa," [25], we argue that some problematic design patterns addressed by code smells or anti-patterns (e.g. cyclic dependency or duplication) may also occur in EA context. Furthermore, the practice of transferring existing concepts into a different domain has been performed to generate first ideas within a new problem domain and inspire further research (e.g. [2], [3]).

Attribute	Meaning
Name	Gives the refactoring solution a meaningful designator
Connected EA Smells	Links the refactoring solution to the targeted EA smells
Derived from	Names the code refactoring solution from which this refactoring solution is derived
Summary	Gives a brief overview of this refactoring solution
Intent	Describes the main goal of this refactoring solution
Motivation	Describes the reasons of applying this refactoring solution
Prerequisites	Describes the conditions prior to applying this refactoring solution
Impact	Describes the influence of this refactoring solution to EA qualities
Mechanics	Describes the practical steps to apply this refactoring solution
Discussion	Describes the situations when this refactoring solution can be useful
Graphical example	Shows a graphical representation of the refactoring solution to reduce misinterpretations

Table 2
Attributes of EA refactoring solutions (adapted from [4], [26], and [27]).

Concept Mapping. The second step in our methodology is to establish a mapping between the concepts in code and EA modelling, which should serve as a basis for conceptually transforming the code refactoring solutions collected into EA refactoring solutions.

Although the code and the EA lie on two opposite sides in the spectrum of abstraction, the languages used for modelling them share a good deal of similar constructs in that both support the description of architectural aspects. Previous studies have proposed some mappings between the notations of UML and ArchiMate, which are the most used languages for code and EA modelling, respectively. Some of these mappings are described in the ArchiMate specification [17]—as some notations thereof are indeed derived from UML [19]—, while the rest have been exclusively suggested by the studies. Since most code refactoring solutions have been described and exemplified using UML, we strongly argue that such mappings can guide the transformation of code refactoring solutions into EA refactoring solutions.

We believe that the first mapping between the concepts in UML and ArchiMate was proposed by Wiering et al. [18]. Their study focuses on identifying matching concepts and relationships by comparing the notations in the two languages by the properties thereof. Another attempt was made in a study by Gill, which focuses on finding concepts in other modelling languages beside ArchiMate which are applicable for EA modelling [16]. The resulting contribution includes a mapping of some concepts in UML to ArchiMate. Furthermore, Lankhorst also advocate the use of ArchiMate in conjunction with other modelling languages (including UML) to create a more holistic EA description [19]. To support this in practice, this contribution highlights not only the similarities but also important differences in ArchiMate and UML, e.g. the absence of separate concepts for *service* and *interface* as well as the reverse interpretation of the ArchiMate relationship *serving* in UML. Lastly, another mapping of UML and ArchiMate is proposed by Gericke, which also considers ArchiMate concepts under the motivation layer [20]. Table 1 summarizes all these mappings which we use as a basis to conduct the concept transformation.

Concept Transformation. To finally derive EA refactoring solutions, the code refactoring solutions collected are processed as follows:

- We analyzed the descriptions and examples of code refactoring solutions to identify instances of UML concepts and relationships. Based on the mapping of UML and ArchiMate notations shown in table 1, the identified UML constructs are translated into ArchiMate constructs. Since each UML notation does not necessarily map exclusively with one ArchiMate notation (e.g. both Collaboration and Interface in UML can be translated into Interface in ArchiMate), the translation has to be inferred rationally from our understanding of what solution is possible and feasible for the EA smell addressed.
- In some cases, the translation may not directly result in an ArchiMate construct that conveys a valid solution in the context of EA. Such ArchiMate construct is either modified for a reasonable EA refactoring solution or excluded from consideration.

Because of the manual concept transformation in this process, the outcome is highly influenced by the skill, knowledge, and experience of the researchers. Also, since the mapping used gives more than one matching ArchiMate notations for every UML notation, the resulting translation may vary depending on the researcher's own understanding and interpretation of the two modelling languages (UML and ArchiMate).

EA smell → EA refactoring solutions	EA smell → EA refactoring solutions
Ambiguous Viewpoint → 5 Viewpoints	Jumble → Architecture Partitioning
Architecture by Implication → Goal Question Architecture	Lazy Component → Ghostbusting or Inline Service
Big Bang → Process Manager	Message Chain → Process Manager or Extract & Move Data Object
Bloated Service → Merge input	Missing Abstraction → Add Abstraction
Business Process Forever → Process Manager	Multifaceted Abstraction → Split Phase
Chatty Service → Front End Gate	Nanoservices → Front End Gate
Combinatorial Explosion → Extract Shared Functionality	No Legacy → Process Manager
Connector Envy → Extract Component	Nothing New → Process Manager
Cyclic Dependency → Cyclic Dependency Removement	Overgeneralization → Extract Shared Functionality
Data-Driven Migration → Functionality First, Data Last	Sand Pile → Grouping
Data Service → Encapsulate Data Service	Scattered Parasitic Function → Merge Components
Dead Component → Remove Dead Component	Shared Persistency → Encapsulate Business Objects
Deficient Encapsulation → Break Up Component	Shotgun Surgery → Grouping
Deficient Names → Rename Component	Stovepipe System → Architecture Framework or EA Planning
Dense Structure → Complexity Reduction	Strict Layers Violation → Move Service to Different Layer
Documentation → Rename Component	Temporary Solution → Extract Temporary Solution
Duplication → Extract Shared Functionality	The God Object → God Object Decomposition
Feature Envy → Move Component or Front End Gate	The Shiny Nickel → Plan Ahead
Golden Hammer → Boundaries	Vendor Lock-In → Isolation Layer
Hub-like Modularization → Break Up Component	Warm Bodies → Small Project
Incomplete Abstraction → Grouping	Weakened Modularity → Split Modularity
Incomplete Node or Collaboration → Introduce Local Extension	Wrong Cuts → Reorganization
Inconsistent Versioning → Semantic Versioning	

Table 3
Catalog of EA Refactoring Solutions

3.4. Demonstration

In this DSR step, the use of the artifact is demonstrated to solve one or more instances of the problem [21]. The fundamental questions to be answered are, "What utility does the new artifact provide?" and "What demonstrates that utility?" [22] In the context of this study, the EA refactoring solutions proposed should provide solution alternatives for mitigating EA smells and the practical details needed to understand their feasibility as well as relevance according to the current conditions. Therefore, to demonstrate the applicability of our result for solving the problems described in section 3.1, we illustrate some small scenarios of mitigating EA smells using the EA refactoring solutions proposed. For this purpose, we use a small ArchiMate model which contains some EA smells. Then, we identify candidates of EA refactoring solutions and select one that we deem the most appropriate for the EA smell given. It is worth noting that the systematic approach to prioritizing EA refactoring solution candidates is still a subject to future work; hence, the selection of EA refactoring solution demonstrated in this paper relies on the authors' perspectives. Finally, we elaborate the architectural changes suggested by the EA refactoring solution selected, show the resulting ArchiMate model after applying these architectural changes, and discuss the impact on the ArchiMate model's overall quality.

3.5. Evaluation

The evaluation step in DSR aims to review the effectiveness of the artifact proposed in supporting a solution to the problem [21]. In this paper, this step is embodied in a discussion, in which we explain how the main RQs of this study have been answered based on our findings and their demonstration. Since the discussion presented in this paper is focused only on the qualitative performance of EA refactoring solution within our example scenario, we recommend future research to extend the evaluation by including more (real-world) examples and quantifiable measures (e.g. using EA model quality framework [28]).

3.6. Communication

Finally, the communication step focuses on presenting the artifact proposed, its utility, and its effectiveness to researchers and other relevant audiences. [21] To communicate our results in an intuitive and consistent manner, we document the EA refactoring solutions in a standard template (see table 2) which we adapted from some existing templates of refactoring solution used in [4] [26] [27]. Furthermore, the EA refactoring solutions are categorized based on the domains of the corresponding EA smells (i.e. business, technology, and application domains) [2] and made publicly available on a web page [29].

Name	Process manager	Encapsulate Business Objects
Description	This refactoring increases the user orientation of service interfaces with the goal to enable user involvement in business processes. A process manager that realizes the service calls is created. Previous caller of the process call the process manager now with process control data that parameterizes the wanted process.	Route data access through dedicated data services that encapsulate the needed business objects.
Connected EA smell	Business Process Forever, Big Bang, Nothing New, No Legacy	Shared Persistency
Derived from	Process Manager	Encapsulate Variable
Intent	Create a process manager component which orchestrates the components involved in the process.	Narrow down the data visibility by routing data access through dedicated data services.
Motivation	This refactoring is meant to create a better user involvement. It also adds more flexibility and extensibility to the architecture.	Reduce the likeability that teams unwillingly depend on each other because the visibility of the data is too broad.
Prerequisites	A long, very strict process chain with many different stakeholders involved.	Multiple business services that access the same database.
Impact	This refactoring makes the application service interfaces more user oriented which enables the user involvement in business processes.	The structure created by the refactoring reduces data visibility to only those of the owned business objects, thereby preventing any unwanted interdependence between teams and services.
Mechanics	1. Generate a new service P called the Process Manager 2. Add a service call from P to all the process components of the Process manager 3. All calls to the individual process components calls now the Process Manager with process control data C that parameterize the wanted original Process. Test each call individually. 4. Remove the service calls between the services that belong now to the Process Manager P	1. Create a new data service that will contain the dedicated data services 2. For each business service, create a dedicated data service that routes the access to the database. Test each data service. 3. Move the database into the grouping data service.
Discussion	This is not only a refactoring but also an architecture pattern itself. It can be used to generate more online involvement for everyone that should be involved. In a peer-to-peer system it is usually not desired to have many centralized services, so the process manager needs to be evaluated carefully.	The data visibility is very narrow when using this refactoring which reduces unwanted dependencies between teams.

Table 4
Documentation of the Process Manager and Encapsulate Business Objects EA refactoring solutions

4. Result

As a result, this study yields a catalog of 37 EA refactoring solutions for all 45 EA smells proposed in [2] (see table 3). In the catalog, some EA refactoring solutions are suggested for multiple EA smells, such as the Process Manager EA refactoring solution which is suggested for the Big Bang, Business Process Forever, Message Chain, No Legacy, and Nothing New EA smells. Such result is obtained because some code refactoring solutions–from which we derived EA refactoring solutions–have also been suggested for mitigating various code smells. While we are aware that specific adaptations may be necessary to make one kind of solution works for different problems, the catalog currently provides only a general description of how to apply each EA refactoring solution. Describing such adaptations is a subject to future work.

Furthermore, the catalog also suggests multiple EA refactoring solution candidates for some EA smells, such as the Move Component and Front End Gate which are suggested as solution candidates for mitigating the Feature Envy EA smell. Such result is obtained because we identified different studies which suggest different code refactoring solutions for the same code smell. While we are aware that the selection of an appropriate EA refactoring solution should consider the specific circumstances surrounding the EA smell problem, such specific circumstances are beyond the scope of this study. Therefore, we suggest future studies in this context to investigate the possibility to systematically prioritize all EA refactoring solutions recommended for a certain EA smell. In such a prioritization approach, various quality requirements may be considered, such as scalability and extensibility.

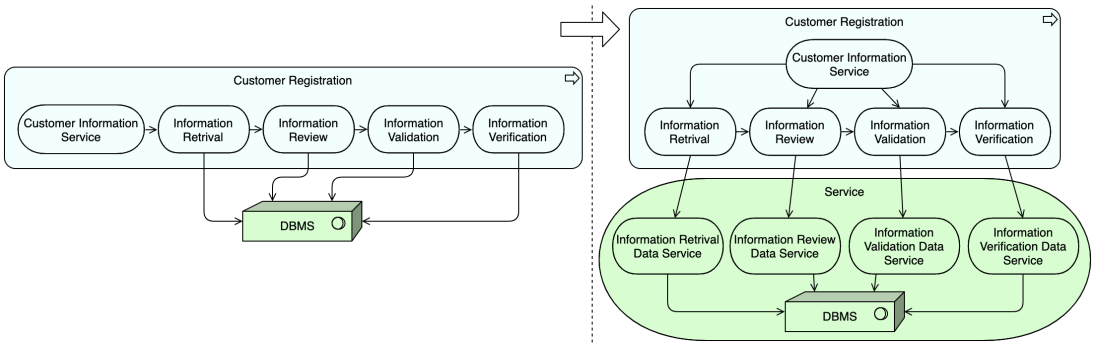

Figure 1: EA refactoring demonstration, showing before (left) and after (right) refactoring EA smells in an ArchiMate model

5. Demonstrating EA Refactoring Solutions

In order to illustrate the application of the proposed EA refactoring solutions, we performed an experiment to apply EA refactoring solutions on a small ArchiMate model (see fig. 1), which we adapted from [2]. This ArchiMate model contains two EA smells: First, a *message chain* due to the sequential calls over 5 application services to fulfill the same application process (i.e. CUSTOMER REGISTRATION). Second, a *shared persistency* because of the direct relations between 3 application services and a database management system (i.e. DBMS).

The first EA smell (i.e. *message chain*) occurs when at least 5 services sequentially interact to fulfill a process, which may harm availability and evolvability [30]. To solve this, our catalog (see table 3) proposes two possible EA refactoring solutions, i.e. the **process manager** (inspired from 'process manager' in [31]) and **extract and move data object** (adapted from 'extract and move class' in [4]). To select from several possible refactoring solutions, the architect must first evaluate whether the main idea and practical details thereof suit the context of the subjected EA smell. In the presented ArchiMate model, we assume that the exemplified *message chain* does not stem from data but process architecture issues. Therefore, the **process manager** is chosen, as presented in table 4.

The chosen refactoring solution suggests to restructure the collaboration scheme between the chained services into an orchestration scheme—in which one service acts as a 'process manager' that coordinates independent services to fulfill the requested business process. This scheme eases the maintenance and extension of the supported business process, thereby remediating the availability and evolvability issues posed by the *message chain*. In our ArchiMate model, this refactoring solution is applied by first encapsulating all involved operations across the chained application services into independent activity modules that are accessible through external interfaces. Afterwards, a process manager must be implemented (in this case, the CUSTOMER INFORMATION SERVICE) which receives requests for customer information processing and fulfills these by delegating tasks to the available services.

The second EA smell (i.e. *shared persistency*) is detected when multiple services access the same data collection or schema, thereby reducing team and service independence [30]. For this, our catalog proposes the **encapsulate business objects** refactoring solution (adapted from 'encapsulate variable' in [4]), as presented in table 4. This refactoring solution suggests to route accesses to the DBMS through dedicated data services; each of which encapsulates all business objects required by one application service. This structure reduces data visibility to only those of the owned business objects, thereby preventing unwanted interdependence between teams and services.

6. Discussion

In this section, we describe the extent to which the main RQs (see section 1) of this study have been answered through our finding and its demonstration; elaborate the implications of the EA refactoring solutions identified for researchs and practitioners; and identify the potential threats to the internal and external validity of our result.

6.1. Explanation of the Result

With regards to answering the RQ1 and its sub-questions, this study identifies the first catalog of EA refactoring solutions for all the EA smells proposed in [2]. We achieved this result by performing a conceptual transformation on relevant code refactoring solutions, which relies on a mapping between code and EA modelling concepts. To document the resulting EA refactoring solutions, we use a template of attributes which we adapted from some existing templates in code refactoring domain. Finally, we demonstrated the use of several refactoring solutions

in some small scenarios of EA smell. While the solution choices demonstrated may not be suitable to all cases in reality, we believe that the presented catalog provides a useful platform for EA researchers and practitioners to develop new or better refactoring solutions to common problems in EA.

6.2. Implication for Practitioners & Researchers

Recent study in EA has proposed the concept of EA debt [1] which represents the divergent EA evolution from the EA standards and principles. One possible source of EA debt is the implementation of sub-optimal solution design. The role of EA smells is to indicate the signs of existing sub-optimal solutions within the IT landscape, so that further investigation can be triggered in time [2].

With regards to answering the RQ2, the implication of our result can be seen from both practical and research perspectives. For EA practitioners, the proposed EA refactoring solutions may help to find possible methods or techniques for solving the EA smells identified. Also, such a solution catalog can help to establish common terminologies within an organization, thereby supporting various stakeholders to discuss possible solutions.

For the EA research community, our result gives motivation and food for thought for further investigations into the concept of EA refactoring. Future attempts to extend the catalog by transforming refactoring solutions from other domains (e.g. process smell, infrastructure smell, etc) may also adopt the conceptual transformation methodology used in this study. To allow public contributions in the development of EA refactoring solutions, the catalog has been published in a website together with a guideline on contributing [29].

6.3. Threats to Validity

The results of this study have to be seen in the light of some limitations. In this section, we present an assessment of possible threats to the internal and external validity of our result. Internal validity focuses on the reliability of the result within the given environment, whereas external validity focuses on the ability to generalize the result [32].

Internal Validity. The design of our methodology may pose certain threats to the internal validity of our result. Firstly, the code refactoring solutions analyzed were collected from a selection of relevant books on code smell and refactoring, thereby threatening the completeness of the EA refactoring solution catalog. Secondly, the mapping between UML and ArchiMate notations used in the transformation was summarized from some selected sources, thereby threatening the completeness of the mapping used in this study. Last but not least, the process of transforming code refactoring solutions into EA domain relies on subjective analysis, which potentially results in biased considerations upon deriving the EA refactoring solutions. To reduce the bias, the resulting EA refactoring solutions were reviewed and decided among the authors of this paper. Despite these weaknesses, we believe that the resulting EA refactoring solution catalog is a step in the right direction towards solutions for common design issues in EA.

External validity. As the resulting EA refactoring solutions are yet to be evaluated in industrial context, their descriptions may still rather hypothetical and theoretical, thereby posing challenges to their adoption. Furthermore, as there could be multiple solution alternatives for every EA smell, further research is still needed to identify possible factors and conditions which influence the suitability of a certain EA refactoring solution. Finally, the adoption of EA refactoring solutions greatly depends on the progress in EA smell research. At the time of writing, the existing EA smells are yet to be evaluated and their descriptions enriched with practical examples.

7. Conclusion & Future Work

Our main goal is to support EA practitioners in analyzing possible improvements with regards to the current circumstances and interests. We strongly argue that approaches to EA improvement must extend from EA modelling capabilities. Therefore, this study investigates the concept of refactoring in the context of EA modelling. Our methodology focuses on defining EA refactoring solutions for the EA smells proposed in [2] by analyzing the known associations between code refactoring solutions and code smells. The resulting contribution is a catalog of EA refactoring solutions which is publicly available on a web page [29].

In spite of this progress, there is still work to be done, especially in improving the catalog, or where further work is necessary. Some potential future works in this context are as follows: Firstly, further EA refactoring solutions can be derived for EA smells from different domains, such as the EA process smells [3]. For such purpose, we suggest to adapt the methodology from the one used in this study, e.g. by integrating a mapping of ArchiMate to another modelling language of interest as proposed in [16]. Secondly, empirical studies can be performed to further review and improve the hitherto knowledge in this area. Last but not least, tool supports can be developed for, e.g., recommending suitable EA refactoring solutions (e.g. [33]) or supporting the implementation thereof.

References

[1] S. Hacks, H. Höfert, J. Salentin, Y. C. Yeong, H. Lichter, Towards the definition of enterprise architecture debts, in: 23rd IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2019, Paris, France, October 28-31, 2019, 2019, pp. 9–16. doi:10.1109/EDOCW.2019.00016.

[2] J. Salentin, S. Hacks, Towards a catalog of enterprise architecture smells, in: N. Gronau, M. Heine, H. Krasnova, K. Poustcchi (Eds.), Entwicklungen, Chancen und Herausforderungen der Digitalisierung: Proceedings der 15. Internationalen Tagung Wirtschaftsinformatik, WI 2020, Potsdam, Germany, March 9-11, 2020. Community Tracks, GITO Verlag, 2020, pp. 276–290. doi:10.30844/wi_2020_y1-salentin.

[3] B. Lehmann, P. Alexander, H. Lichter, S. Hacks, Towards the identification of process anti-patterns in enterprise architecture models, in: H. Lichter, S. Aydin, T. Sunetnanta, T. Anwar (Eds.), Proceedings of the 8th International Workshop on Quantitative Approaches to Software Quality, co-located with 27th Asia-Pacific Software Engineering Conference (APSEC 2020), Singapore (virtual), December 1, 2020., CEUR-WS.org, 2020, pp. 47–54. URL: http://ceur-ws.org/Vol-2767.

[4] M. Fowler, Refactoring - Improving the Design of Existing Code, Addison Wesley object technology series, Addison-Wesley, 1999. URL: http://martinfowler.com/books/refactoring.html.

[5] A. S. Nyamawe, H. Liu, Z. Niu, W. Wang, N. Niu, Recommending refactoring solutions based on traceability and code metrics, IEEE Access 6 (2018) 49460–49475. URL: https://doi.org/10.1109/ACCESS.2018.2868990. doi:10.1109/ACCESS.2018.2868990.

[6] W. G. Griswold, Program Restructuring as an Aid to Software Maintenance, Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, USA, 1992. UMI Order No. GAX92-03258.

[7] S. J. Thompson, Refactoring functional programs, in: V. Vene, T. Uustalu (Eds.), Advanced Functional Programming, 5th International School, AFP 2004, Tartu, Estonia, August 14-21, 2004, Revised Lectures, volume 3622 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 331–357. URL: https://doi.org/10.1007/11546382_9. doi:10.1007/11546382_9.

[8] A. Folli, T. Mens, Refactoring of UML models using AGG, Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 8 (2007). URL: https://doi.org/10.14279/tuj.eceasst.8.112. doi:10.14279/tuj.eceasst.8.112.

[9] J. Reimann, C. Wilke, B. Demuth, M. Muck, U. Aßmann, Tool supported OCL refactoring catalogue, in: M. Balaban, J. Cabot, M. Gogolla, C. Wilke (Eds.), Proceedings of the 12th Workshop on OCL and Textual Modelling, Innsbruck, Austria, September 30, 2012, ACM, 2012, pp. 7–12. URL: https://doi.org/10.1145/2428516.2428518. doi:10.1145/2428516.2428518.

[10] M. Misbhauddin, M. Alshayeb, Uml model refactoring: A systematic literature review, Empirical Software Engineering 20 (2015) 206–251. URL: https://doi.org/10.1007/s10664-013-9283-7. doi:10.1007/s10664-013-9283-7.

[11] M. R. Hacene, S. Fennouh, R. Nkambou, P. Valtchev, Refactoring of ontologies: Improving the design of ontological models with concept analysis, in: 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 2, IEEE Computer Society, 2010, pp. 167–172. URL: https://doi.org/10.1109/ICTAI.2010.97. doi:10.1109/ICTAI.2010.97.

[12] D. Silingas, E. Mileviciene, BPMN 2.0 Handbook, Future Strategies Inc., Lighthouse Point, FL, USA, 2012, pp. 125–134. URL: http://www.futstrat.com/.

[13] S. W. Ambler, P. J. Sadalage, Refactoring Databases: Evolutionary Database Design, Addison-Wesley Professional, 2006.

[14] J. Bruel, M. Mazzara, B. Meyer (Eds.), Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6-8, 2019, Revised Selected Papers, volume 12055 of *Lecture Notes in Computer Science*, Springer, 2020. URL: https://doi.org/10.1007/978-3-030-39306-9. doi:10.1007/978-3-030-39306-9.

[15] M. Lippert, S. Roock, Refactoring in large software projects: performing complex restructurings successfully, John Wiley & Sons, 2006.

[16] A. Q. Gill, Agile enterprise architecture modelling: Evaluating the applicability and integration of six modelling standards, Inf. Softw. Technol. 67 (2015) 196–206. URL: https://doi.org/10.1016/j.infsof.2015.07.002. doi:10.1016/j.infsof.2015.07.002.

[17] The Open Group, Archimate 3.1 specification, 2019. URL: https://pubs.opengroup.org/architecture/archimate3-doc/.

[18] M. J. Wiering, M. M. Bonsangue, R. van Buuren, L. Groenewegen, H. Jonkers, M. M. Lankhorst, Investigating the mapping of an enterprise description language into UML 2.0, Electronic Notes in Theoretical Computer Science 101 (2004) 155–179. URL: https://doi.org/10.1016/j.entcs.2004.02.020. doi:10.1016/j.entcs.2004.02.020.

[19] M. M. Lankhorst (Ed.), Enterprise Architecture

at Work - Modelling, Communication and Analysis, Fourth Edition, Springer, 2017. URL: https://doi.org/10.1007/978-3-662-53933-0. doi:10.1007/978-3-662-53933-0.

[20] T. Gericke, ArchiMate to UML Mapping, Technical Report, Adocus AB, Stockholm, Sweden, 2018. URL: http://www.adocus.com/media/21703/archimate-to-uml-mapping-whitepaper.pdf.

[21] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, J. Manag. Inf. Syst. 24 (2008) 45–77. URL: http://www.jmis-web.org/articles/765.

[22] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Q. 28 (2004) 75–105. URL: http://misq.org/design-science-in-information-systems-research.html.

[23] S. Kebir, I. Borne, D. Meslati, Automatic refactoring of component-based software by detecting and eliminating bad smells - A search-based approach, in: L. A. Maciaszek, J. Filipe (Eds.), ENASE 2016 - Proceedings of the 11th International Conference on Evaluation of Novel Approaches to Software Engineering, Rome, Italy 27-28 April, 2016, SciTePress, 2016, pp. 210–215. URL: https://doi.org/10.5220/0005891602100215. doi:10.5220/0005891602100215.

[24] W. H. Brown, R. C. Malveau, H. W. S. McCormick, T. J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, 1st ed., John Wiley & Sons, Inc., USA, 1998.

[25] F. A. Fontana, V. Lenarduzzi, R. Roveda, D. Taibi, Are architectural smells independent from code smells? an empirical study, Journal of Systems and Software 154 (2019) 139–156. URL: https://doi.org/10.1016/j.jss.2019.04.066. doi:10.1016/j.jss.2019.04.066.

[26] G. Suryanarayana, G. Samarthyam, T. Sharma, Refactoring for Software Design Smells: Managing Technical Debt, 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2014.

[27] J. Kerievsky, Refactoring to patterns, in: C. Zannier, H. Erdogmus, L. Lindstrom (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004, 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004, Proceedings, volume 3134 of *Lecture Notes in Computer Science*, Springer, 2004, p. 232. URL: https://doi.org/10.1007/978-3-540-27777-4_54. doi:10.1007/978-3-540-27777-4_54.

[28] S. Hacks, F. Timm, Towards a quality framework for enterprise architecture models, EMISA Forum 38 (2018) 31–32.

[29] L. Liss, H. Kämmerling, P. Alexander, H. Lichter, Enterprise architecture refactorings, 20.10.2021. URL: https://swc-public.pages.rwth-aachen.de/ea-refactoring-solutions/web-catalog/.

[30] J. Salentin, S. Hacks, Enterprise architecture smells, 2020. URL: https://ba-ea-smells.pages.rwth-aachen.de/ea-smells/.

[31] J. Král, M. Zemlicka, The most important service-oriented antipatterns, in: Proceedings of the Second International Conference on Software Engineering Advances (ICSEA 2007), August 25-31, 2007, Cap Esterel, French Riviera, France, IEEE Computer Society, 2007, p. 29. URL: https://doi.org/10.1109/ICSEA.2007.74. doi:10.1109/ICSEA.2007.74.

[32] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, Experimentation in Software Engineering - An Introduction, volume 6 of *The Kluwer International Series in Software Engineering*, Kluwer, 2000. URL: https://doi.org/10.1007/978-1-4615-4625-2. doi:10.1007/978-1-4615-4625-2.

[33] H. Zhang, S. Jarzabek, A bayesian network approach to rational architectural design, Int. J. Softw. Eng. Knowl. Eng. 15 (2005) 695–718. URL: https://doi.org/10.1142/S0218194005002488. doi:10.1142/S0218194005002488.