Tool Support for Learning Architectural Guidance Models from Architectural Design Decision Models

Amirali Amiri, Evangelos Ntentos
University of Vienna, Faculty of Computer Science, Software Architecture Group, Vienna, Austria
University of Vienna, Doctoral School Computer Science, Vienna, Austria
firstname.lastname@univie.ac.at

Uwe Zdun
University of Vienna, Faculty of Computer Science, Software Architecture Group, Vienna, Austria
uwe.zdun@univie.ac.at

Sebastian Geiger
Siemens, Vienna, Austria
sebastian.a.geiger@siemens.com

ABSTRACT

This paper presents an approach to architectural knowledge management that does not assume existing architectural design decisions or pattern applications are documented as architectural knowledge, but benefits from more existing data. We drew inspiration from manual qualitative research methods for mining patterns and architectural knowledge and created a guideline model of which the ADD models are instances. We evaluated our approach on 11 cases from the gray literature. We found that it can provide suitable recommendations after modeling only a single case and reaches theoretical saturation and recommendations with low to very low errors after only 6-8 cases. Our approach shows that creating a reusable architectural design space is possible based only on limited case data. Our approach not only provides a novel approach to architectural knowledge management but can also be used as a tool for pattern mining.

CCS CONCEPTS

• **Software and its engineering → Patterns**; *Designing software*;

KEYWORDS

Architectural Knowledge Management, Architectural Design Decisions, Design Patterns

ACM Reference Format:
Amirali Amiri, Evangelos Ntentos, Uwe Zdun, and Sebastian Geiger. 2023. Tool Support for Learning Architectural Guidance Models from Architectural Design Decision Models. In *28th European Conference on Pattern Languages of Programs (EuroPLoP 2023), July 5–9, 2023, Irsee, Germany*. ACM, New York, NY, USA, Article 4, 14 pages. https://doi.org/10.1145/3628034.3628037

1 INTRODUCTION

Architectural Knowledge Management (AKM) [15, 34] is about the management of (software) Architectural Knowledge (AK), often in the form of Architectural Design Decisions (ADDs). Today, these ADDs, i.e., the significant design decisions in a software system, are seen as an integral part of the architecture of a software system [14]. An ADD includes, among other things, the decision to make, a context in which the decision is made, a set of decision drivers or forces that influence the decision outcome, a set of alternative options considered, and a decision outcome (i.e., the chosen option).

Decision options are closely related to patterns. In software architecture decision-making, decisions for or against time-proven solutions often have patterns as (chosen or alternative) options [10, 41]. Recurring decision outcomes are good starting points for pattern mining in fields where such patterns still need to be mined and documented.

Early AKM and ADD tools [3, 27, 34, 35] have focused on ADD and AK capturing. Later, suggestions to improve steps in the AKM process were proposed, and tool support was provided. Reusable ADDs [40, 41] have been proposed and tool-supported [21, 22] to minimize the work of ADD documentation across projects in which similar ADDs are recurring. Patterns can play a crucial role in such approaches, as they are time-proven solutions already well-documented in the literature [10, 41]. Other suggestions to improve AKM process steps are, for example, to support AK retrieval via Web search [32], consider Technical Debt during AKM [2, 31], use ontologies to organize and manage the AK [8] or make recommendations based on the requirements [20]. Please note that while each of these approaches solves a specific problem of AKM, each tool still requires substantial manual work to be applied in a project. For instance, the reusable ADD approaches require effort invested into maintaining the reusable ADD model to reuse it across projects, a Web retrieval tool would require constant review and editing, and an AK ontology needs to be maintained.

We observed in industry projects which introduced ADD tools [21] that initially, practitioners are interested and see the need for such tools. But unfortunately, in many industry settings, under daily time pressure, it is hard to find volunteers who feel responsible for maintaining a common AK base, and it isn't easy to find incentives for such work. Consequently, in today's industry practice, AK is often recorded with simple templates, e.g., in Markdown [18, 26] with no further AKM tool support than storing the template in a version repository, or even not at all. To address this problem, we set out to study how far it is possible to benefit from AKM tool support across different projects without a need to perform extra maintenance work for the common AK base.

EuroPLoP 2023, July 5–9, 2023, Irsee, Germany

ACM ISBN 979-8-4007-0040-8/23/07.
https://doi.org/10.1145/3628034.3628037

A possible approach for a tool to learn AKM is through machine learning. Some existing machine-learning-based AKM approaches focus on classification or adaptation tasks only, and can only learn AKM from past projects to make it reusable [1, 4, 24, 33, 38]. To create such approaches, a substantial amount of data from past ADDs in a particular field or pre-trained machine-learning models are required. However, for new fields, such as the Cyber-Physical Systems (CPS), knowledge bases for even widely used software patterns do not exist. Informal sources, such as heterogeneous blog posts, system documentation, and open-source implementations, exist as knowledge sources for new fields. Therefore, an approach that requires no existing AKM data but improves as more data is present is needed.

For these reasons, the researchers took inspiration from the manual, qualitative research methods used to mine patterns and architectural knowledge. Specifically, they drew from the more informal Pattern Mining approach used in the pattern community [30] and the Grounded Theory research method [7, 9]. In their approach, different projects in a company or community are added to the design space and modeled as ADD Models. Their tool uses this information to gradually build up a Guidance Model, or a reusable ADD Model, of which the ADD Models are instances. An Guidance Model is a set of guidelines that provides direction, best practices, and recommendations for designing and implementing software architectures. It aims to assist software architects and development teams in making informed decisions. The system learns the Guidance Model and can start making recommendations based on frequency-based or text similarity recommendations relatively early. The only required maintenance work needs to be done during modeling if an architect creates a name conflict. The architect can review it immediately and resolve it for the whole design space. Although no interaction between projects is strictly necessary, some interaction between architects would be helpful, such as agreeing on central decisions or option names.

We evaluated our approach using 11 cases from the gray literature and compared various frequency-based, and text-similarity recommendation approaches. We reached theoretical saturation and recommendations with low to very low errors in our cases after only 6-8 cases. Even after modeling a single case, our tool provided already fitting recommendations in many cases. This confirms that our approach can help to create a reusable ADD design space based only on limited (and potentially incomplete) instance data.

Please note that a secondary goal of our approach is supporting pattern mining. Each instance in a design space modeled with our tool can be seen as a known use of several pattern candidates (the chosen options). The decision option types in the Guidance Model are thus the pattern candidates. They are modeled along with contexts, forces, and related patterns and practices.

This paper is structured as follows: In Section 2, we discuss the research methods pattern mining and GT used as our background, as well as the research methods used in the evaluation. Next, in Section 3 we introduce our approach, and then illustrate our prototype tool in Section 4. The evaluation is presented in Section 5, and our contributions are compared to related work in Section 6. Finally, we conclude in Section 7.

2 BACKGROUND AND RESEARCH METHODS

Our approach is inspired by the pattern mining methods in the pattern community. The Grounded Theory (GT) research method [7, 9] is a systematic research method for discovering theory from data. Iterative steps are taken when interpreting data, whereby the focus and central goal is to build a theory grounded in the data. Data analysis should occur during data collection and not afterward.

The most important activity in GT is Constant Comparison: the researcher continuously and iteratively compares pre-existing data and concepts with new data. Any newly-arising abstract concepts should then be compared with pre-existing concepts and data. The concepts are organized into categories, so-called codes, and are compared and linked to properties and each other via relations [7]. The concepts, categories, and properties derived from the data should guide the next iteration of research activities. Theoretical sampling involves actively seeking out new data based on the results of the previous iteration, considering the kinds of data that should be collected next [16]. This is continued until Theoretical Saturation is reached, i.e., "the point in category development at which no new properties, dimensions, or relationships emerge during analysis" [7].

We applied the methodology of Strauss and Corbin [7], which is characterized by three types of coding activity:

- *Open coding* involves developing concepts based on the data sources. It entails asking specific (and consistent) questions about the data, precise (and consistent) coding, and memo writing with minimal assumptions.
- *Axial coding* is the development of categories and the linking of data, concepts, categories, and properties.
- *Selective coding* refers to the integration of the categories that have been developed and their grouping around a central core category.

Hentrich et al. provide details on how GT's coding process is mapped to pattern mining [11]. Riehle et al. [30] explain various such systematic pattern mining methods, and propose steps for discovering, codifying, evaluating, and validating the patterns during pattern mining. In GT-based pattern mining, those steps are embodied in GT's coding and constant comparison processes.

We evaluated our approach using 11 cases documented in the gray literature. These were modeled as different projects in a company or a community. We passed the names for decisions, decision options (patterns and practices), external solutions, decision contexts, and forces used in the sources in our tool. We compared them to the recommendation for the reusable decision names selected during the modeling. This way, it is possible to show that after only 6-7 cases, simple frequency-based recommendations contain the selected value with a very low error but could perform better in suggesting the correct value or ranking it in the first three suggested values. Text similarity recommendations for a keyword phrase reach very low or low errors not only for suggesting the correct value but also for predictions, suggesting on correct value, and ranking it in the first three suggested values after only 7-8 cases. We compared predictions based on Word2Vec-based angular similarity and Levenshtein similarity. Both perform close to each other; we thus suggest using the simpler Levenshtein similarity in practical applications. As the number of new elements in the

models can be seen as a measure of Theoretical Saturation, and it seems Theoretical Saturation is approached after around 8 cases in our evaluation where only a very few elements are newly introduced, it seems that our approach's recommendation performance correlates well with the Theoretical Saturation.

3 APPROACH

In this section, we first introduce the main steps of our approach and then explain the meta-models of the ADD models and the guidance model. Based on these foundations, we explain examples of both models from our cases used in the evaluation. Next, we describe the relations of our approach to pattern mining. Finally, we explain our recommendation approaches.

3.1 Main Steps

The process of our approach is illustrated in Figure 1, which follows the main coding steps of GT. To facilitate the mapping, we have used the same terminology as GT. Our approach assumes that multiple models will be created in a shared design space that pertains to a particular domain or topic. For instance, in our study, all cases were related to the interaction between IoT devices and sensors, edge devices, and the cloud in a CPS. The projects collect data in various forms such as field notes, source code notes, documentation, annotations, and memos, which are then modeled in our tool to identify new possible ADD model elements. This manual identification process is akin to Open Coding in GT.

Once a new element is identified, it can be added to the tool, and our system automatically recommends a model element based on prior usage frequency or a text similarity recommendation if a name is provided. The user can either accept or reject the suggestion and decide to use the new element as a new type. If a new type is added, the guidance model is updated automatically by incorporating the new element or relation type. These steps correspond to the Axial Coding step in GT, where categories and relations of concepts are identified. In this step, we refer to it as semi-automatic because the process of adding newly identified elements to the tool is performed manually by the user. However, the recommendations and updates of types within the tool are automated, requiring no manual intervention.

Finally, our tool allows for selective coding where architects can detect and resolve inconsistencies, errors, or simplify the model by renaming type names. For example, two concepts might refer to the same decision option but have different names, which can be unified by renaming one or both to a common name. Our tool automatically updates the guidance model accordingly, making it easy for users to rename model entities or types. This step is also categorized as semi-automatic as the user manually performs modifications such as renaming, while the tool automatically handles the updates for similar elements.

3.2 Meta-model of the ADD Model

Figure 2 shows the meta-model for ADD models. Each ADD model is described in our tool as a *Model*. A model has a name, a description, and a link to the *Guidance Model*. This guidance model is automatically derived from this ADD model and all other ADD models in the design space, as explained above.

The model contains the *Decisions* of the ADD model. The decision has a *Context*, which is described by a domain object that denotes the system part or aspect in which the decision is applied. Each decision has chosen *Options* and considered alternative *Options*. All options are *Solutions*. In addition to decision options, the model also contains external *Solutions*. These solutions are needed to describe the model fully but are outside the core decisions.

An option has *Forces*, which can have a force impact. We model the impact using a five-point Likert scale: ++ for very positive impact, + for positive impact, *o* for neutral impact, − for negative impact, and −− for very negative impact.

Finally, decisions, solutions, and options can have *Relations*. A solution can relate to another solution, but the relation's source or target must be an option. All solutions in the model must be linked directly or via other options to a decision. Decisions and options can have next-decision relations, too.

The following constraints formally define the possible relation sources and targets:

```
context Solution
inv: self.relations->forAll(r | r.source.oclIsKindOf(Option) or
  r.target.oclIsKindOf(Option))
context Option
inv: self.next_decision_relations->forAll(r | r.target.oclIsKindOf(Decision))
context Decision
inv: self.next_decision_relations->forAll(r | r.source.oclIsKindOf(Decision) and
  r.target.oclIsKindOf(Decision))
```

Decisions, contexts, solutions (and options), forces, and relations are all named elements, meaning they can have an optional name, an optional description, and must have a type (described in the next section).

3.3 Meta-model of the Guidance Model

Figure 3 shows the details of guidance models as a meta-model. This part of our meta-model is linked to the meta-model from Figure 2 using type and model relations. It contains mainly the types that are learned from and then shared by multiple instance models. Like the model, the *Guidance Model* has a name and a description. But unlike the names and descriptions on *Named Element Type*, these are type names and descriptions. In addition, the guidance model has a list of all models derived from it.

Context Type, *Decision Type*, *Option Type*, *Solution Type*, *Relation Type*, and *Force Type* correspond to the ADD meta-model elements starting with the same base name. These elements and their relations are learned from the instances. Thus, they have very similar relations and attributes. Chosen and alternative options are not distinguished at the type level, as a chosen option of one instance can be the alternative option of another or vice versa. The following constraints express the implied type relations formally:

```
context Named_Element
inv: self.oclIsKindOf(Decision) implies self.type.oclIsKindOf(Decision_Type)
inv: self.oclIsKindOf(Relation) implies self.type.oclIsKindOf(Relation_Type)
inv: self.oclIsKindOf(Force) implies self.type.oclIsKindOf(Force_Type)
inv: self.oclIsKindOf(Context) implies self.type.oclIsKindOf(Context_Type)
inv: self.oclIsKindOf(Option)  implies self.type.oclIsKindOf(Option_Type)
inv: self.oclIsKindOf(Solution) implies self.type.oclIsKindOf(Solution_Type)
```

Solutions (and thus options) have a solution that, at the moment, is either Practice, Pattern, or Do Nothing. Do Nothing is a special kind that indicates that this solution (or option) requires no action to be realized. Practice is a generic solution, and Pattern implies a Practice that has formally been described as a recurring Design Pattern in the literature.

The same kinds of relation types as relations are supported. The relation types thus require the same constraints:

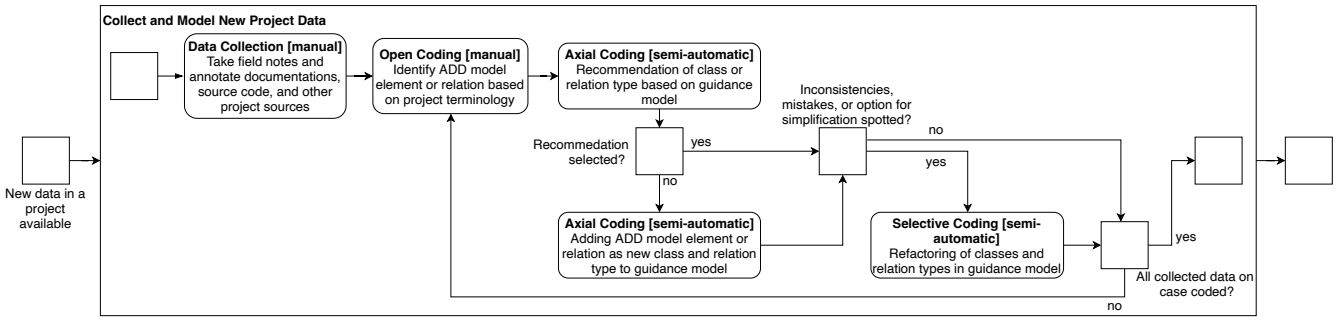


Figure 1: Overview of steps of the approach

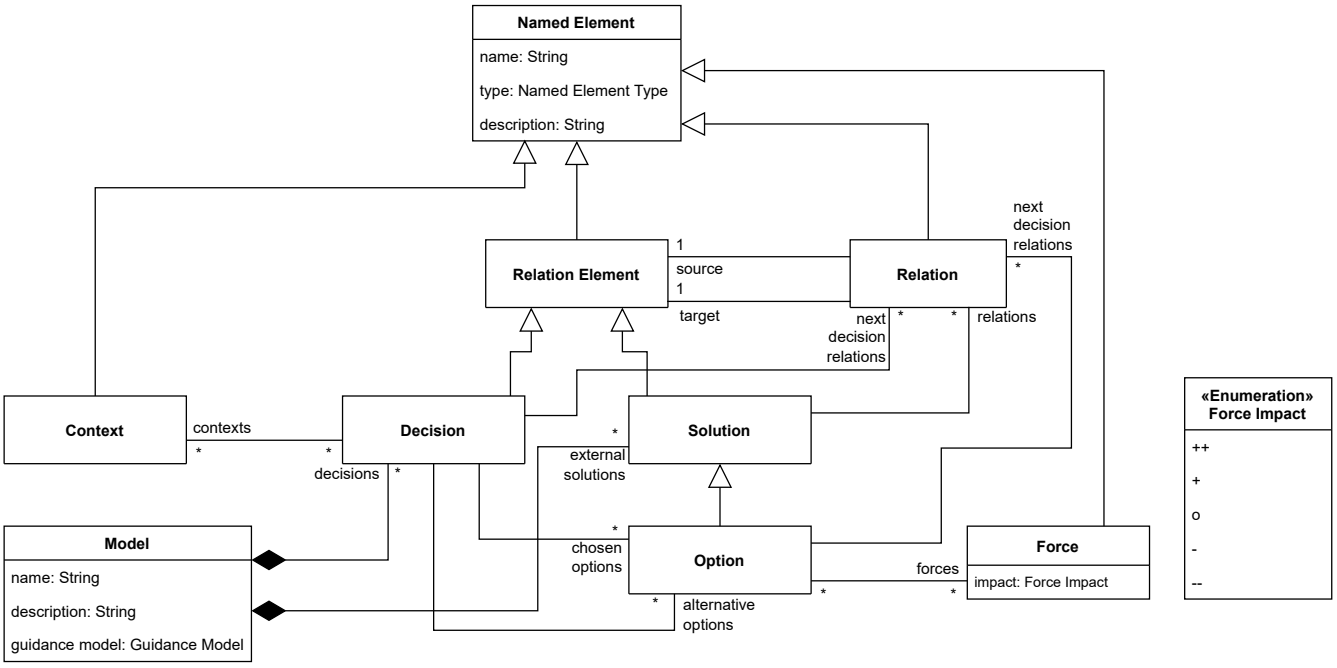


Figure 2: Meta-model for ADD Models

```
context Solution_Type
inv: self.relation_types->forAll(r | r.source_type.oclIsKindOf(Option_Type) or
  r.target_type.oclIsKindOf(Option_Type))
context Option_Type
inv: self.next_decision_relation_types->forAll(r |
  r.target_type.oclIsKindOf(Decision_Type))
context Decision_Type
inv: self.next_decision_relation_types->forAll(r |
  r.source_type.oclIsKindOf(Decision_Type) and
  r.target.oclIsKindOf(Decision_Type))
```

Each relation type can have a stereotype that indicates the kind of relation. As solution-to-solution relations (including option relations) have different relation kinds to next decision relations, we model the stereotype as a string here and model the possible relation types in the following constraints:

```
context Solution_Type
inv: self.relation_types->forAll(r | Set{'Requires', 'Uses', 'Can Use',
  'Can Be Combined With', 'Can be Realized By', 'Has Variant', 'Extension',
  'Is-a', 'Realizes', 'Includes', 'Can Include', 'Alternative To',
  'Rules Out', 'Influences', 'Leads To', 'Enables'}->includes(r.stereotype))
context Option_Type
inv: self.next_decision_relation_types->forAll(r | Set{'Mandatory Next',
  'Optional Next', 'Next', 'Consider If Not Decided Yet'}->includes(
r.stereotype))
context Decision_Type
inv: self.next_decision_relation_types->forAll(r | Set{'Mandatory Next',
  'Optional Next', 'Next', 'Consider If Not Decided Yet'}->includes(
    r.stereotype))
```

3.4 ADD and Guidance Model Examples

To illustrate our approach, let us first discuss two ADD model excerpts from the AIE and THW cases used below in our evaluation. Each excerpt shows a single decision instance from these ADD models. The decision instance in Figure 4 shows a decision for either using MQTT/AMQP-based messaging or a Device Protocol for Handling the IoT Traffic on the edge when integrating using the Microsoft Azure IoT Edge platform. This decision must be made in a Device and an Edge Component context. We can select one of the options for each device/edge component combination. Here, two chosen options are possible: a Direct Connection with Messaging

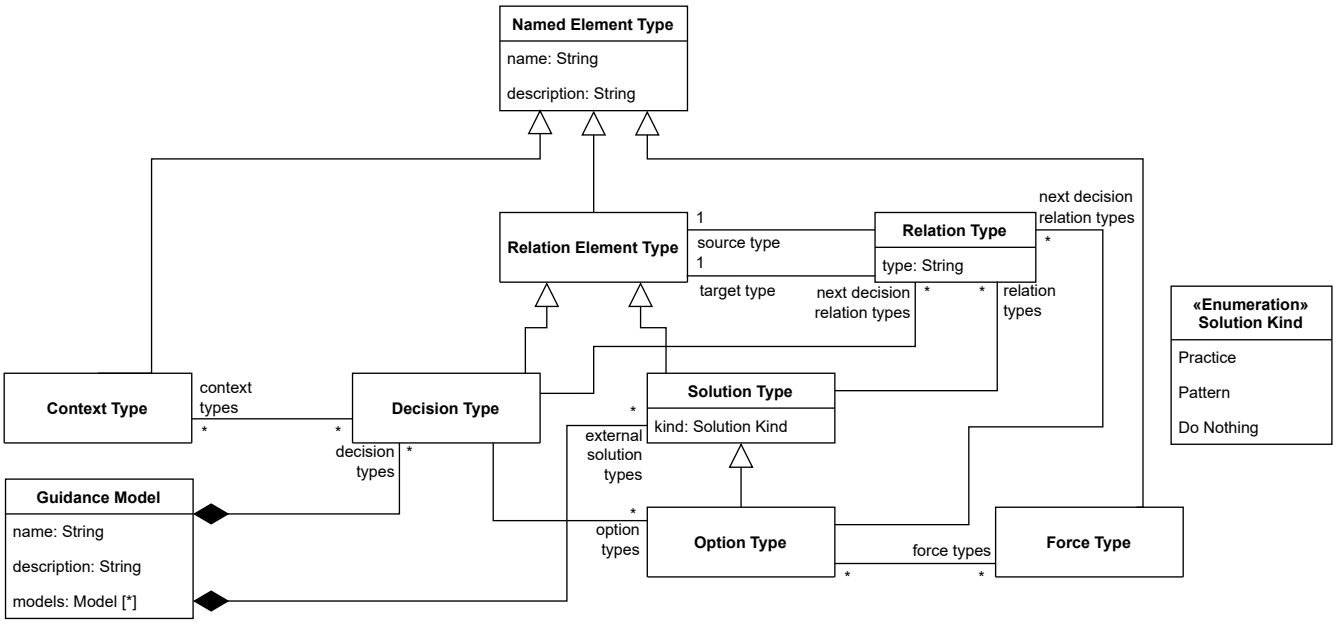


Figure 3: Meta-model for Guidance Models

using the said messaging protocols (if messaging is supported on the device) or using a Device Protocol for the connection and then integrating via the Messaging Gateway pattern [12]. A possible alternative option would be to Directly Connect with Device Protocol, but this is not recommended to be used in this case. As two options can use a Device Protocol, a solution (option) from another decision, we modeled this option-to-option relation here.

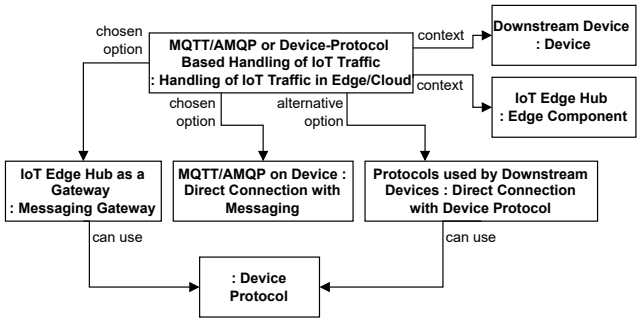


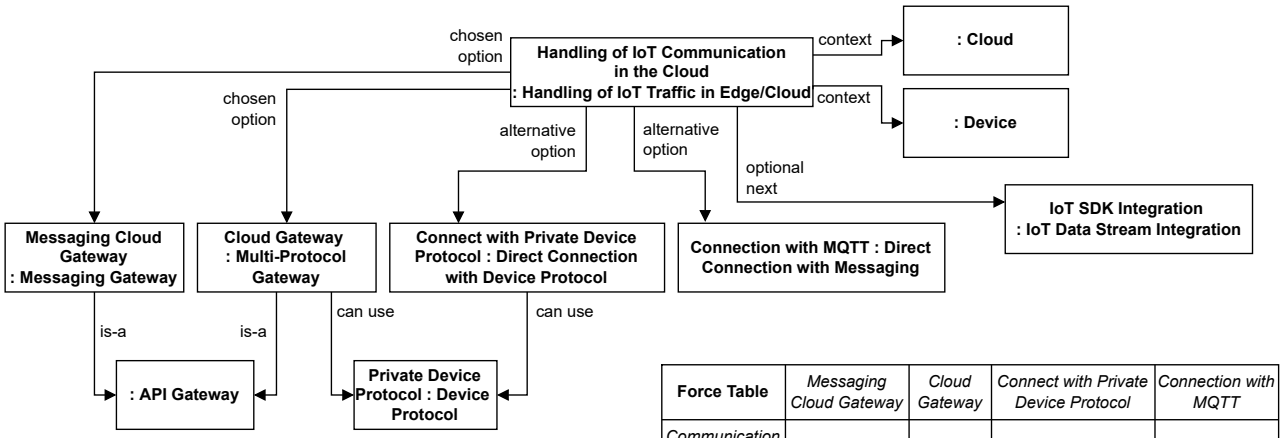
Figure 4: Example ADD Model Excerpt: Handling of IoT Traffic in the AIE case

The second excerpt from the THW case in Figure 5 models a more complex variant of the same decision, representing best practices documented by the company Thoughtworks. The two chosen options are a Multi-Protocol Cloud Gateway or the Messaging Gateway pattern. Both have an is-a relation to the pattern API Gateway [29], which is an external solution (i.e., a solution that is not an option in any decision of this design space but is still modeled to explain the relationship between the two options of this decision). As alternatives, the Direct Connection with Device Protocol (called Private Protocol in this case) and the Direct Connection with Messaging (called Connection with MQTT in this case) was considered. Here, the combination of any Device and the Cloud needs to be considered as the decision context, as this case is about direct device-to-cloud integration.

Three forces are modeled, which are possible decision drivers for this decision: Communication Efficiency, Maintainability, and Development Effort with the respective force impacts. We depicted them in a separate table instead of drawing the UML relations to these forces to make the figure readable. Finally, a next decision relation is modeled: An IoT Data Stream Integration decision using an IoT SDK is discussed.

Figure 6 shows the guidance model learned from these and the other cases. As can be seen, all discussed decisions, options, forces, solutions, contexts, and relations (and a few more) reappear here as respective types. We show in orange the classes and relations learned from the AIE case, and in purple, the classes and relations learned from the THW case. As can be seen, all model elements, except for two classes and two relations, have already been learned after only those two cases.

As the learning is based on type names (strings), it is important that users check the resulting guidance model after each modeling step for inconsistencies, mistakes, and so on and take refactoring steps during selective coding if required. A common mistake is that a user needs to realize that a type is already in use. For instance, suppose that in the THW case, the user has yet to realize that Connect with Private Device Protocol is an instance of Direct Connection with Device Protocol. Consequently, this new option would appear in the guidance model alongside the existing Direct Connection with Device Protocol. A simple *rename option type(Connect with*



Force Table	*Messaging Cloud Gateway*	*Cloud Gateway*	*Connect with Private Device Protocol*	*Connection with MQTT*
Communication Efficiency	++	+	- -	-
Maintainability	++	+	-	-
Development Effort	o	-	- -	-

Figure 5: Example ADD Model Excerpt: Handling of IoT Traffic in the THW case

Private Device Protocol, Direct Connection with Device Protocol) refactoring would solve this issue. Our tool would automatically find the existing type and update all instance model relations using Connect with Private Device Protocol to link to the existing option type.

3.5 Relations to Pattern Mining

Our approach generates a guidance model and statistics of the frequency of use of each model element type in the model. Suppose an option or solution is not a pattern documented in the literature. In addition, it is recurring in the design space and is frequently suggested as a chosen option. In that case, this option is a new pattern candidate in the domain of the design space.

In our experience, it also makes sense to closely inspect a pattern candidate's related chosen options, alternative options, and solutions. Of course, a related frequently used chosen option might be another pattern candidate in the simplest case. But as patterns are modeled in more depth than ADD design spaces, related chosen options might be variants of the same patterns and thus should become part of the newly described pattern. Similarly, related external solutions might become parts of the pattern's solution description. Related alternative options that are never used as a chosen option might become a section in the pattern describing non-solution or might be described as anti-patterns. Both require thorough research in the literature and existing systems to confirm the non-solution or anti-pattern status.

Typically, the contexts of options and solutions used in a pattern are candidates to describe the pattern context and the forces for the pattern forces. The decision with its option relations, as well as the inter-decision relations, option relations, and solution relations can either become part of the pattern's solution or pattern relations.

3.6 Recommendations of Model Element Types

When users of our tool encounter new potential concepts to be added to an ADD model, it would be helpful for the efficient and effective creation of the model to recommend the model element type based on what our tool has learned as a guidance model so far. Here, for each type of model element (decisions, contexts, options, solutions, forces), we can encounter essentially two different situations if the guidance model already contains concepts searched for by the user: The user has not enough information yet to name a potential model element and wants a recommendation from the guidance model. Or the user has a name for a model element, but it might be differently phrased than the model element type in the guidance model the user is searching for.

Frequency-based text recommendation is a straightforward recommendation that works even when the user cannot provide any information. It is a method of measuring the similarity between two texts based on the frequency of common words. This approach is simple to implement and can be effective in certain situations. However, if the user has some information, such as a keyword phrase (model element name), frequency-based text recommendation is likely inferior to text similarity recommendation approaches. *Text similarity recommendation* is a common task in natural language processing and information retrieval, and there are several approaches to measure the similarity between two texts. Two commonly used methods are Levenshtein similarity and Word2Vec-based cosine similarity.

The *Levenshtein similarity* [39] is a measure of similarity between two texts based on the minimum number of single-letter changes (insertions, deletions, or substitutions) required to transfer one text to the other. The Levenshtein distance is defined as:

$$levenshtein\_similarity(t_1, t_2) = \min_{\pi \in AP(t_1, t_2)} \sum_{(i,j) \in \pi} [i \neq j]$$

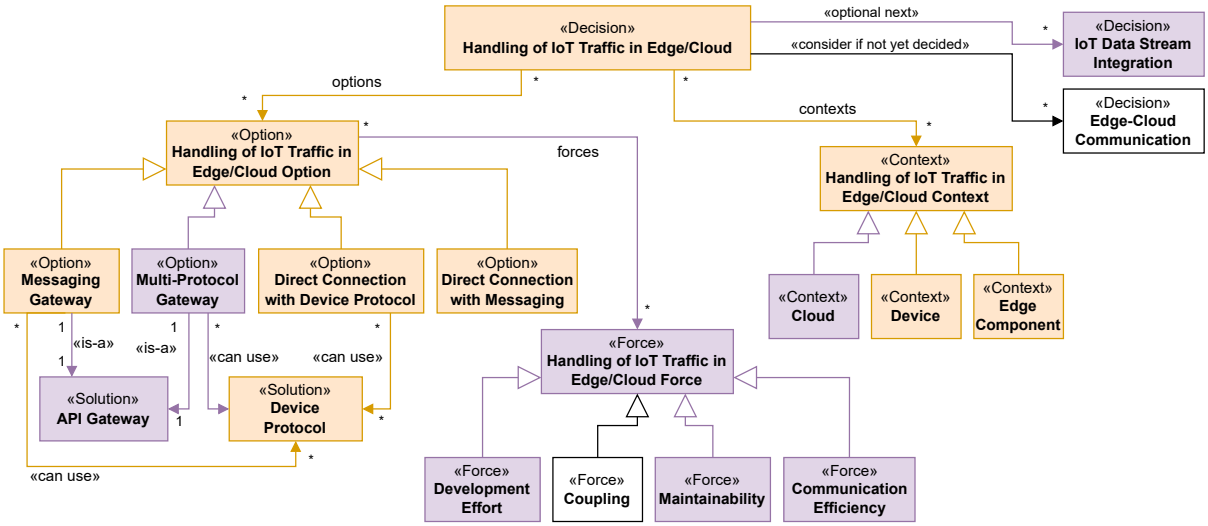


Figure 6: Example Guidance Model Excerpt: Handling of IoT Traffic at the Edge or in the Cloud

where $t1$ and $t2$ are the two texts being compared, $AP(t_1, t_2)$ is the set of all possible alignment paths between the two texts, and the sum is over all pairs (t_1, t_2) in the alignment path π.

Word2Vec-based cosine similarity [23] is a measure of similarity between two texts based on the cosine similarity between the word embeddings of the words in the texts. Word embeddings are vector representations of words that capture the semantic meaning of the words. The cosine similarity between two vectors is a measure of the angle between the vectors, where a value of 1 indicates that the vectors are perfectly aligned. A value of 0 indicates that the vectors are orthogonal. In Word2Vec, cosine similarity is calculated as the dot product of the vectors representing the two words divided by the product of their magnitudes (L2 norm). The dot product is a measure of the similarity between two vectors, while the magnitudes represent the lengths of the vectors.

$$cosine\_similarity(\mathbf{v}_1, \mathbf{v}_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \, \|\mathbf{v}_2\|}$$

There is a problem with using cosine similarity directly for our purposes. A cosine similarity of -1 means that the vectors are diametrically opposed (opposite). A value of 0 means that the vectors are orthogonal (right-angled). However, if a design space contains opposite and orthogonal values, we want the orthogonal value to be ranked lowest and the opposite values to be relatively high. For example, if the design space contains "No Digital Twin" options, "Digital Twin" options, and numerous unrelated options, both the correct and negative options should be ranked higher than the unrelated options when searching for "Digital Twin." We convert the cosine similarity to the *Word2Vec-based Angular Similarity* to achieve this. The angular similarity is more intuitive in our context and, like Levenshtein similarity, has values ranging from 0 to 1. This conversion was also used by Cer et al. [6], who found that using angular distance-based similarity performed better on average than raw cosine similarity. The formula for this is:

$$angular\_similarity(\mathbf{v}_1, \mathbf{v}_2) = \frac{\arccos(cosine\_similarity(\mathbf{v}_1, \mathbf{v}_2))}{\pi}$$

We used the Spacy library [13] in Python as Word2Vec implementation with the medium-sized English model trained on written Web texts en_core_web_md. Smaller, more efficient models did not work well in our context, as they contained few specialized software engineering terms used in our Cyber-physical Systems design space.

The Levenshtein similarity and the Word2Vec-based Cosine similarity are helpful approaches for measuring the similarity between texts. Which one is more appropriate will depend on the specific requirements and goals of the task. In our evaluation, we test and compare both approaches and the simple frequency-based recommendations for making recommendations in our context. Please note that the performance Levenshtein similarity vs. Word2Vec is not apparent because the more sophisticated Word2Vec approach has some issues in our case: Firstly, sometimes there are relatively short phrases (like some forces or contexts that are just one word long) and, secondly, there are phrases that might be missing as a work embedding in the used pre-trained Word2Vec model-based, e.g., on English language training data sets. For instance, the force Configurability appeared in none of the pre-trained models.

4 TOOL ARCHITECTURE

Figure 7 shows the architecture of our prototype tool. The web frontend of the tool is implemented in React[1] as a single-page application. The frontend communicates with a RESTful service written in Python Flask[2]. This service handles the incoming requests using the Codeable Models backend[3]. Codeable Models is a Python tool for specifying meta-models, models, and model instances in code.

[1] https://reactjs.org/
[2] https://flask.palletsprojects.com/en/2.2.x/
[3] https://github.com/uzdun/CodeableModels

We used CodeableModels to define meta-models for components and relationships. We also developed automated constraint checkers and PlantUML[4] code generators to create graphical representations of all of our meta-models and model instances.

Codeable Models aims to provide an easy-to-use Application Programming Interface (API) for coding software design models akin to UML models. This API comprises our Guidance Metamodel, Visualization Generator, Evaluation scripts, etc. For persistence, we use MongoDB[5] with the PyMongo driver[6]. Using the Docker technology[7], the tool is containerized into three containers, i.e., frontend, backend, and persistence.

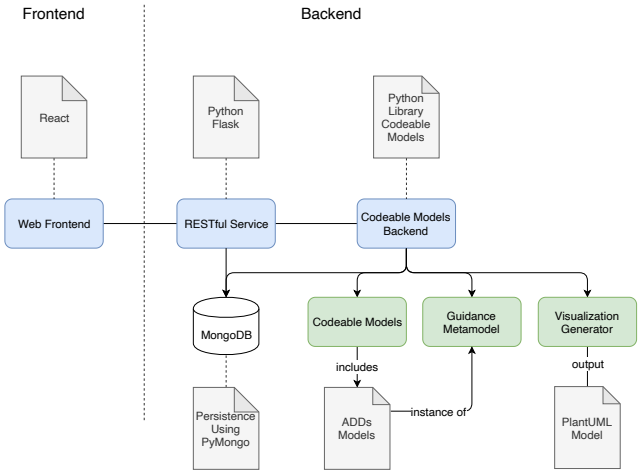


Figure 7: Tool Architecture Diagram

Figure 8 shows the flow of our tool. A user accesses the frontend using a Web User Interface (UI). The Web UI communicates to the backend to create, read, update and delete model elements. Recommendations are provided to the user when creating or updating elements. The Codeable Models Generator takes the Guidance Metamodel as input and creates ADD models. In case the user requests for visualizations, these ADD models are given to Visualization Generator that outputs PlantUML models.

5 EVALUATION

To evaluate our research, we studied the following research questions:

- **RQ1:** Can theoretical saturation of the guidance model be achieved after modeling a small number of ADD models? If so, when can it be expected?
- **RQ2:** How well does recommendation work **a)** when the user provides no data and **b)** when (parts of) keywords from the ADD model cases are provided? **c)** Which of the proposed recommendation approaches delivers the best results?

Please note that RQ1 and RQ2 are linked, as a good recommendation after a few cases can be seen as an indicator of theoretical saturation.

[4]https://plantuml.com/
[5]https://www.mongodb.com/
[6]https://www.mongodb.com/docs/drivers/pymongo/
[7]https://www.docker.com/

5.1 Case Selection

The search engines of Google, Bing, and DuckDuckGo served as the primary source for *case studies selection*. We followed the recommendations by Petersen et al. [28] (originally for systematic mapping studies) for establishing search phrases according to the PICO principles [17]. We mainly applied two PICO principles, *population* and *intervention*. Overall, we considered 45 cases. Then, we applied inclusion/exclusion rules. In particular, we excluded cases that were too brief, not written by practitioners, not technically detailed enough to derive ADDs, or mainly contained biased or product advertisement information. In the end, we selected the 11 cases in Table 1 to be included in our evaluation. The cases were chosen independently and in parallel to our approach's development.

5.2 Evaluation Methods

After selecting the cases, we carefully modeled each as an ADD model in the order in Table 1. The table also summarizes the sizes of the ADD models we have modeled for each case. We used the terms from the cases as ADD model element names and derived generic type names from those model element names. Whenever a new name emerged, we compared it to existing names and decided if the type name needed to be refactored. Our tool used new elements observed in the ADD models to learn the guidance model automatically. In the first five cases, we frequently needed to change the type names given before; in the following 7 cases, this happened only rarely. In our evaluation, we used the ADD model element names as keyword phrases to recommend the type names used in the guidance model. For each single model element in each of the twelve cases, we calculated the following recommendations:

(1) A recommendation purely based on the frequency of the model elements. The model element with the highest frequency is recommended first, then the second highest frequency, then the third, and so on. We used the frequency-based text recommendation method, as explained in Section 3.6.
(2) A recommendation based on the keyword phrase entered by the user. Here, we use the one used as ADD model element name.
(3) A recommendation based only on the first word of the keyword phrase entered by the user to estimate how well the approach works if only incomplete information is entered yet.
(4) A recommendation based only on the first three characters of the keyword phrase entered by the user to estimate how well the approach works if only very incomplete information is entered yet.

For the text-based recommendations, we calculated the Levenshtein similarity and the angular similarity based on Word2Vec, as explained in Section 3.6. We evaluated this for each of the 11 cases consecutively, for each of the model element types listed in Table 1, i.e., Columns 3-7 and 9 of the table. Please note that Column 8, i.e., Relations, is not an element type on its own in the evaluation. Instead, the source and target of each relation must be predicted (either a solution, option, or decision prediction).

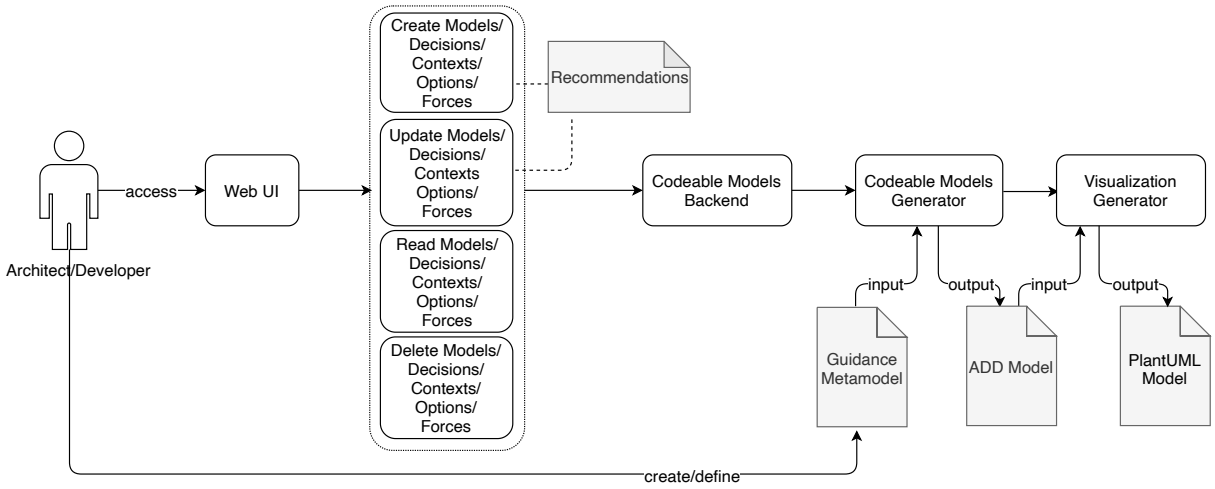


Figure 8: Process flow architecture of the prototype

Table 1: Overview of the Inspected Cases

ID	Title/URL	(#) Decisions	(#) Contexts	(#) Options	(#) Forces	(#) Solutions	(#) Relations	(#) Model Elements
CRO	*How to Build an Industrial IoT Project Without the Cloud.* URL: https://www.iiot-world.com/industrial-iot/connected-industry/how-to-build-an-industrial-iot-project-without-the-cloud/	2	1	8	7	0	1	19
AIE	*Understand the Azure IoT Edge runtime and its architecture.* URL: https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime?view=iotedge-2020-11	10	5	25	20	0	13	73
THW	*Connecting IoT devices to the cloud.* URL: https://www.thoughtworks.com/insights/blog/iot/connecting-iot-devices-cloud	3	3	10	27	1	11	55
SOL	*Real-time Data Streaming in IoT: Why and How.* URL: https://solace.com/blog/real-time-data-streaming-in-iot/	5	3	12	24	0	4	48
AWS	*Edge to Twin: A scalable edge to cloud architecture for digital twins.* URL: https://aws.amazon.com/de/blogs/iot/edge-to-twin-a-scalable-edge-to-cloud-architecture-for-digital-twins/	8	5	16	0	0	9	38
RHT	*Understanding edge computing for manufacturing.* URL: https://www.redhat.com/en/topics/edge-computing/manufacturing	5	3	12	36	0	4	60
HUS	*Connected Things Without a Cloud.* URL: https://husarnet.com/iot	3	3	5	12	0	0	23
TRI	*How to use Digital Twins for IoT Device Configurations.* URL: https://tributech.io/blog/digital-twins-for-IoT-device-configurations	9	6	16	20	2	12	65
MFL	*Mainflux 0.11 — Digital Twin, MQTT Proxy And More.* URL: https://medium.com/mainflux-iot-platform/mainflux-0-11-digital-twin-mqtt-proxy-and-more-46bde98635fe	7	6	11	0	1	4	29
PRS	*Connecting OPC UA Publisher to Amazon AWS IoT with MQTT.* URL: https://www.prosysopc.com/blog/aws-iot-mqtt-demo/	5	5	9	8	0	3	30
DAW	*Dataworks: Internet Of Things.* URL: https://www.dataworks.ie/iot-a-step-by-step-guide-on-how-to-connect-devices-to-the-cloud/	8	5	13	9	0	11	46

In the context of RQ1, we use the *number of new model element types and relation types introduced per case* as a measure of theoretical saturation. This is a reasonable measure, as theoretical saturation is achieved when the researcher can no longer identify new concepts or relationships in the data. Our models represent new concepts and relationships as new model element types and relation types.

After each case, we calculated two Error Measures, the Mean Absolute Error (MAE) and the Mean Squared Error (MSE). Those are two common error measures used in statistical analysis. The MAE is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the true value of the i-th sample, $\hat{y}_i$ is the predicted value, and n is the total number of samples.

The MSE is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Both the MAE and MSE are used to evaluate the performance of a model, with the MAE being a measure of the average magnitude of the error and the MSE being a measure of the average squared magnitude of the error. The MSE is generally more sensitive to large errors than the MAE, as the errors are squared in the MSE.

For each recommendation, we measured for each case the errors of four values:

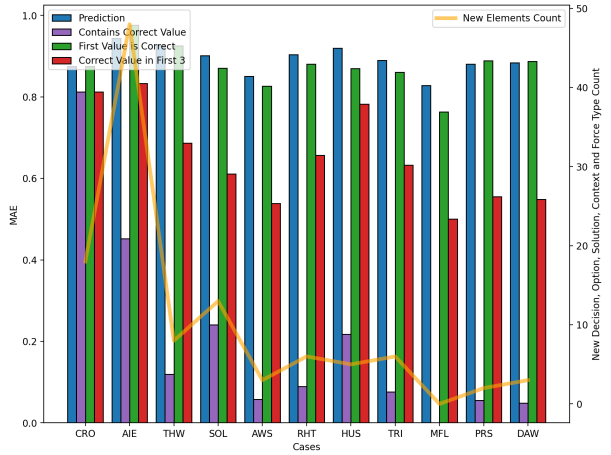


Figure 9: Recommendation based only on model element frequency for all model elements

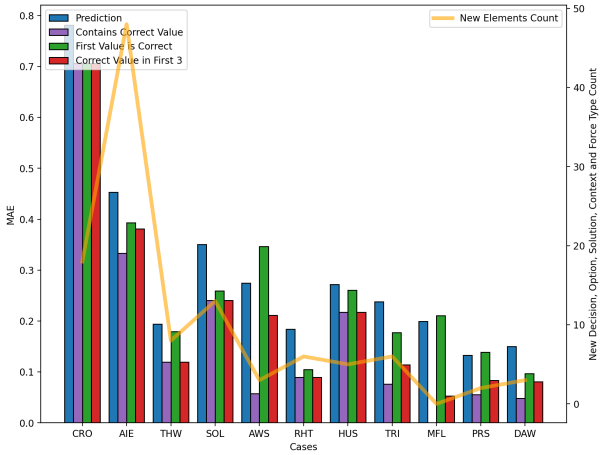


Figure 10: Recommendations with Levenshtein similarity for all model elements after entering the full keyword phrase

- *Prediction:* The error based on the prediction value is calculated as the used similarity measure, i.e., the angular similarity based on Word2Vec or the Levenshtein similarity. We use this prediction value also to create a ranked list of recommendations (ordered from best prediction to worst one). This is used in the following three measures.
- *Contains Correct Value:* This computes the error based on a boolean value (0 or 1) which is 1 if the ranked list of all recommendations contains the correct value and 0 if not.
- *First Value is Correct:* This computes the error based on a boolean value (0 or 1) which is 1 if the ranked list of all recommendations contains the correct value in its first place and 0 if not.
- *Correct Value in the First 3:* This computes the error based on a boolean value (0 or 1) which is 1 if the ranked list of all recommendations contains the correct value in its first three places, and 0 if not.

Please note that here "correct value" refers to the design concept the user wants to model or the model element type the user is searching for. The list of recommendations contains all model element types the system has learned so far. For example, the user searches for the force "Configuration," and the correct value is "Configurability" learned before in the guidance model. Forces are recommended based on the list of all forces that the system has learned so far.

5.3 Results and Discussion

We carefully inspected the results for each case and the two errors. While there are some specific, interesting details, overall, the results are relatively consistent for both errors and across the different kinds of model elements (i.e., Columns 3-7 and 9 of Table 1). Thus, for space reasons, out of our 120 evaluation results investigated, we only report the 7 evaluation results for all model elements (i.e., Column 9 of Table 1) and only the MAE-based ones.

The yellow line "New Elements Count" shown in Figures 9-15 is identical in all figures. The initial cases introduced numerous new model element types (roughly 10-50). After that, only a few new model element types are added per case. There seems to be yet another drop from the tenth case onwards for the last three cases, and <4 new model element types are introduced per case. This indicates that theoretical saturation is likely reached around 5-9 cases, depending on how one likes to set the bound for it. We can conclude for **RQ1** that theoretical saturation can be reached or approached by modeling a few cases, which indicates that our approach is applicable in industrial settings, where multiple similar projects need to be modeled.

The bar chart in Figure 9 shows the recommendation results MAE for all model elements based only on the frequency of their previous appearance. As can be expected, frequency alone shows poor performance for the prediction (blue bars) and recommends the correct values as the first value (green bars). The correct value is more likely contained in the first three values (red bars), but still, the error is high. However, we can see that the correct value is part of our recommendations (purple bars) with a very low error from around 8 cases onwards and a reasonably low error after only modeling 3 cases. This confirms the results on **RQ1** that, after a few cases, it can be expected that our approach has learned the relevant design concepts in design space (aka theoretical saturation is reached). It also shows that, for **RQ2a**, we can conclude that if the user provides no information on what shall be modeled, the best we can expect is a long list of all types that at least contains very likely the correct value.

The bar chart in Figure 10 shows how the MAE changes between cases when the recommendation is made based on the Levenshtein similarity and the whole keyword phrase encountered in the source is entered by the user. As can be seen, all MAE values drop in close correlation to the New Elements Count. As expected, the MAE for the correct value being part of our recommendations (purple bars) drops quickly below an MAE of 0.3 and, after 8 cases, stays below 0.1. The prediction value (blue bars) drops quickly, after 4 cases, below 0.3 and reaches values below 0.2. After 8 cases, the correct

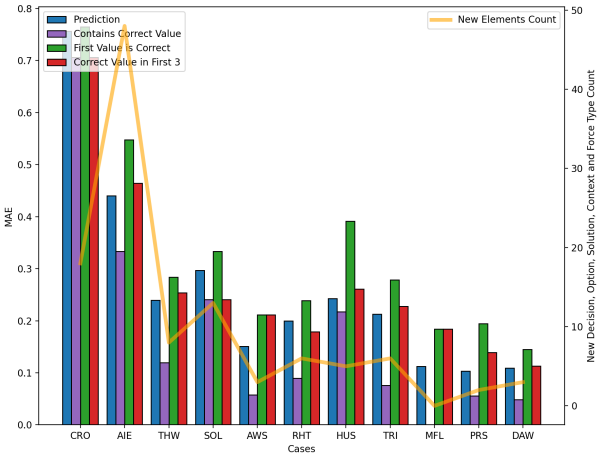


Figure 11: Recommendation with Word2Vec-Based angular similarity for all model elements after entering the full keyword phrase

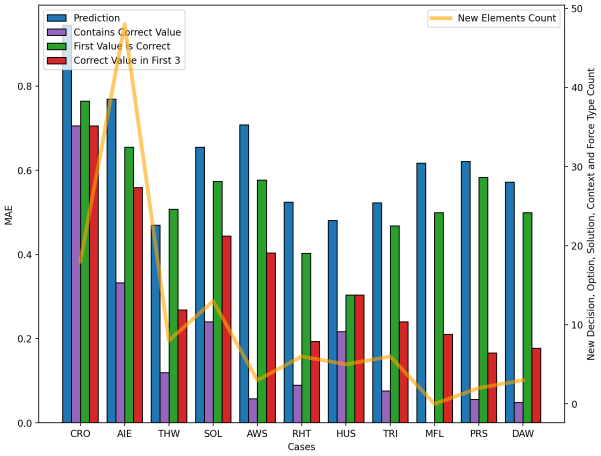


Figure 12: Recommendations with Levenshtein similarity for all model elements after entering the first keyword

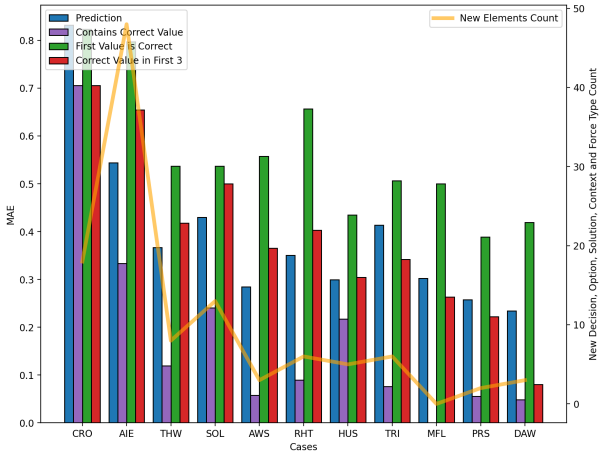


Figure 13: Recommendation with Word2Vec-Based angular similarity for all model elements after entering the first keyword

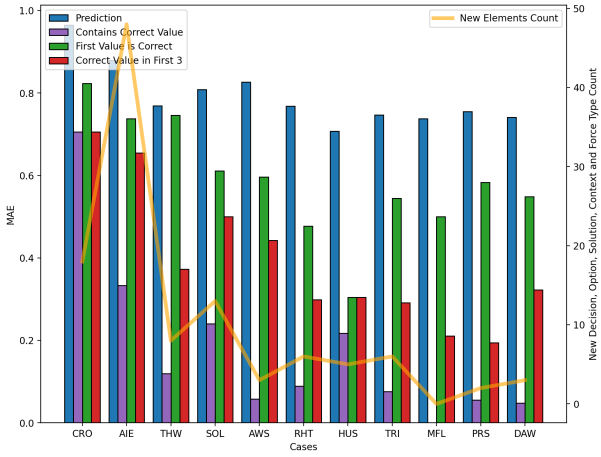


Figure 14: Recommendations with Levenshtein similarity for all model elements after entering the first 3 characters

values are very likely in the first three recommended values (red bars, MAE below 0.1) and likely even in the first value (green bars, MAE below 0.2).

The bar chart in Figure 11 shows the same measurement for Word2Vec-based angular similarity. Overall it shows a very similar performance to the Levenshtein similarity. Still, the predictions (blue bar charts) are a little better, while the other recommendations seem to be a little worse.

The bar chart in Figure 12 shows how the MAE changes when prediction is made based on the Levenshtein similarity and only the first keyword of the keyword phrase encountered in the source is entered by the user. Here, it can be seen that again the recommendation very likely contains the correct value (purple bars, after 7 cases MAE below 0.1) and likely contains the correct value in the first three recommendations (red bars, MAE around 0.2 after 8 cases). But the prediction error (blue bars) and error for a correct first value (green bars) are relatively high. Interestingly, the same measures for the Word2Vec-based angular similarity shown in Figure 13 perform much better for the prediction values, and errors around 0.2-0.3 can be reached after around 8 cases. The errors in predicting the correct value as the first value are similar, but even after 11 cases, the error is still around 0.4.

The bar chart in Figure 14 shows a reasonably low MAE for Levenshtein similarity when only the first 3 characters of a keyword phrase, i.e., minimal information, are entered for the correct value being in the total list of recommendation (purple bars) and also for predicting the value in the first 3 recommendations (red bars). The prediction error and error of the correct value being recommended first (green bar) are both high, but at least there is a chance of about 50% that the correct value is the first recommended. The same

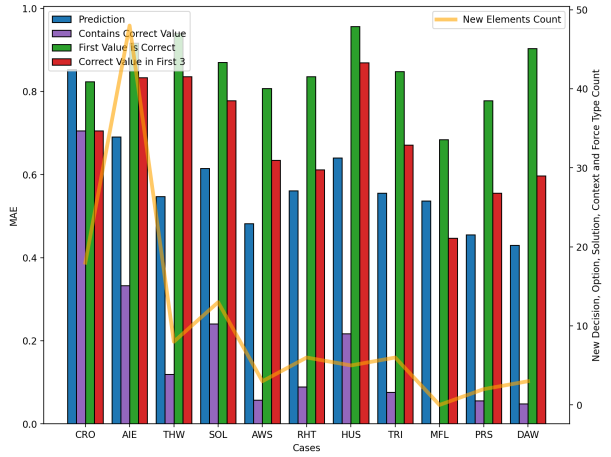


Figure 15: Recommendation with Word2Vec-Based angular similarity for all model elements after entering the first 3 characters

Word2Vec-based recommendation offers lower prediction errors (blue bars) but performs worse in all measures based on the ranked list.

We can conclude for **RQ2b** that both text-based recommendation methods offer excellent performance if the user enters the keyword phrase in the source and likely deliver the correct value as the first recommendation. Still, there is an excellent chance of having the correct value high in the ranked recommendation list if only the first word of the keyword phrase is entered. Yet, even if only the first three chars are provided, this is enough to have a reasonable chance (e.g., around 50%) to have the correct value ranked high.

Finally, for **RQ2c**, we must assert that frequency-based text recommendations perform poorly but are helpful when the user has entered nothing and wants to look at the available options. Then he gets the most likely options ranked first and can be relatively sure that the correct result is contained in the recommendations. Levenshtein similarity and the Word2Vec-based recommendation perform similarly in the evaluation cases, with the Word2Vec-based recommendation offering slightly more precise prediction results, but Levenshtein similarity offers superior ranked recommendations. As Levenshtein similarity is simpler and less expensive to compute, unless the precise prediction value is needed for a task, Levenshtein similarity seems better suited.

5.4 Threats to Validity

Threats to validity refer to factors that may affect the validity or reliability of the results of a study. In the present study, several potential threats to validity should be considered. We discuss the threats to validity based on the threat types by Wohlin et al. [37].

Construct Validity. Construct validity concerns the accurate representation of the intended construct by a measurement. The experience and search-based procedure for finding knowledge sources may have introduced bias. However, this threat is mitigated to a large extent by the chosen research method, which requires additional sources corresponding to the inclusion and exclusion criteria, not a specific distribution of sources. Our procedure is in this regard rather similar to how interview partners are typically found in qualitative research studies in software engineering. The threat remains that our procedures introduced unconscious exclusion of certain sources. We mitigated this threat by assembling an author team with many years of experience in the field, and performing very general and broad searches.

Internal Validity. Internal validity concerns factors that affect the independent variables concerning causality. To increase internal validity we used practitioner reports produced independently of our study. The sample size in this study was small, and few models were used to measure theoretical saturation and to learn the guidance model. This may limit the generalizability of the results to other models or contexts. However, it is unlikely that hundreds of cases can be modeled in an industrial setting such as the one we are interested in. With this in mind, we evaluated our research in a realistic setting.

External Validity. External validity concerns threats that limit the ability to generalize the results beyond the experiment. The study's results may not be generalizable to other contexts or populations because the study was conducted with a specific set of models and data.

Conclusion Validity. Conclusion validity concerns factors that affect the ability to conclude the relations between treatments and study outcomes. The measurement of theoretical saturation may be subject to error because it is based on two proxy measures, the number of new element types in the guidance model and the improvement in error measures. This may affect the accuracy of the results, as the actual theoretical saturation may not be accurately measured.

Because the guidance model was learned from multiple models, there may be omitted variable bias in this study. This may affect the accuracy of the model's recommendations. Overall, it is essential to consider these potential threats to validity when interpreting this study's results and applying the advisory model's recommendations. Further research is needed to address these threats and confirm the validity and generalizability of the results. For example, the study could be replicated for other design spaces.

6 RELATED WORK

This section discussed relevant related works on AKM history, the relation to patterns, AKM tools, reusable AK, and machine-learning-based AK approaches.

6.1 AKM History and Relation to Patterns

This section outlines and compares relevant related works. At the core, the approach presented in this paper is an Architectural Knowledge Management (AKM) tool focusing on AKM discovery and pattern mining. In the 2000s, AKM and ADDs became a hot topic in software architecture research, and senior architects in the industry began to share their AKM practices with the public. Jansen's work [15] set the scene, and Kruchten proposed a taxonomy of ADDs in software-intensive systems [19]. Tyree and Akerman [36] took inspiration from the IBM e-business Reference Architecture (that came with pre-filled ADD records) and motivated why ADDs matter in an article that also presented a rich ADD template.

Zimmermann [40] investigated whether ADD issues recur when similar designs are used on multiple projects and whether decision trees can be mined from the gained experiences. He suggested a method to systematically identify the need for ADDs and the available options in requirements and style definitions. Service-Oriented Architecture (SOA) served as an exemplary architectural style. The ISO/IEC/IEEE 42010 standard [14] recommends capturing decision rationale and gives scoping and filtering advice as well as clarifies many other concepts.

These concepts are closely related to pattern mining. Harrison, Avgeriou, and Zdun [10] introduced the notion of using patterns as options in architectural design decisions, which later was combined with the approach of Zimmermann [41], introducing the notion of Reusable ADDs. Since about 2010, ADDs have made it into the industry project mainstream. For instance, Nygard received a lot of attention with his Architecture Decisions Records (ADR) blog post [26]. The architecture documentation template arc42 [8] dedicates Section 9 to ADDs and gives nine related tips. Kopp [18] suggested MADR as another ADD template in Markdown [9].

6.2 AKM Tools and Reusable AK

Several scientific studies proposed AKM tools. A survey on early tools can be found in [34]. PAKME [3] is a knowledge repository allowing architects to register design decisions to prevent architecture knowledge vaporization. Tofan et al. [35] developed a Web-based tool [10] to help architects in capturing tacit knowledge and architectural decisions. The architectural decisions considered by this tool are related to selecting patterns, technologies, or decomposing systems. Parmar et al. [27] presents an approach to capture ADDs based on templates and to make decisions based on scenarios and non-functional requirements. In these tools, however, no AK reuse happens, and the authors do not use past AK to recommend ADDs.

Several scientific studies investigated further how to reuse AK. Lytra et al. [21, 22] proposed a reusable architectural decision metamodel for quality-driven decision support and the CoCoAdvise tool. Reusable ADDs need to get documented by users and can then be reused in the concrete ADDs of many different projects. The supportive effect of the reusable decision models in decision-making and documentation was tested in two controlled experiments [22]. In [25], the authors reported on an in-depth qualitative study of existing practices in the industry for data management in microservice architectures. That is, this study mined ADDs and patterns from the gray literature to create a reusable ADD model.

Many other sources to enhance AKM tools have been proposed in the scientific literature. Soliman et al. [32] explore the retrieved architectural knowledge (AK) from the Web, and the effectiveness of web search engines, when performing three Attribute-Driven Design architectural steps. To achieve this goal, the authors conducted an exploratory study with software engineers who used Google to find AK for making design decisions. Ampatzoglou et al. [2] provide a cost-benefit approach and supporting tooling that treats architectural decisions like financial investments. In [31], the authors explore technical debt-incurring architectural design decisions in practice. In particular, they focus on the main types of debt-incurring architectural design decisions, why and how they are incurred in a software system, and how practitioners deal with these types of design decisions. de Graaf et al. [8] are proposing ArchiMind and have investigated modified ontologies and have shown how ontological support can be beneficial for the efficient retrieval of architecture knowledge. Silva et al. [20] present a Web-based tool that supports architects by recommending a suitable architectural style based on the system's requirements, particularly the system's quality attributes. The tool encompasses both trade-off resolution over quality attributes and recommendation of architectural styles based on quality attributes. While the listed works follow various approaches to enrich the AKM, so far, all approaches require gathering additional knowledge or modeling to improve certain AKM steps, such as search/retrieval of AK or AKM recommendations.

6.3 Machine-Learning-Based Approaches

Recently, several machine-learning approaches have been proposed for AKM. Ali et al. [1] proposed a supervised machine learning-based approach to classifying architectural knowledge into predefined categories: analysis, synthesis, evaluation, and implementation. In [38], the authors advocate for using machine learning to refine the approach and reveal new patterns of architectural integrity violations. Muccini et al. [24] present a machine learning-based proactive decision-making tool named ArchLearner, for aiding architectural adaptation. In [33], the authors propose that new architectural design practices might be based on machine learning approaches to better leverage data-rich environments and workflows. Bhat et al. [4] proposes a two-phase supervised machine learning-based approach to first automatically detect design decisions from issues and then automatically classify the identified design decisions into different decision categories. In [5] Bhat et.al. introduced a tool called ADeX that helps automate the process of curating design decisions and aids architects and developers in the decision-making process. So far, however, none of these approaches can learn AKM from past projects, but classification or adaptation work is supported.

7 CONCLUSION

This work demonstrates that architectural knowledge management tools can create a reusable ADD design space based on limited data. The simple learning approach we propose does not require existing ADD data to work and can improve as more data is available. We evaluated the approach using 11 cases from the gray literature and found that our tool can provide appropriate recommendations in many cases, even after modeling a single case. Our research shows that AKM tools can help architects create and evolve a common AK base without investing substantial time into curating and maintaining the AK base. Further, creating and evolving such AK bases could be seen as a systematic approach to pattern mining in the field. If our tool proposal would get widely applied in the field, it could improve the situation that no systematic documentation of ADD instances exists. The lack of such data of reasonable quality is one of the main reasons many current machine-learning approaches cannot be applied in our research context. Thus, in the future, our

[8] https://arc42.de/
[9] https://adr.github.io/madr/
[10] https://github.com/danrg/RGT-tool

tool could contribute to changing this situation and creating data sets that would enable machine learning-based approaches.

Acknowledgments.

This work was supported by FFG (Austrian Research Promotion Agency) project MODIS, no. FO999895431.

REFERENCES

[1] Mubashir Ali, Husnain Mushtaq, Muhammad Babar Rasheed, Anees Baqir, and Thamer Alquthami. 2021. Mining software architecture knowledge: Classifying stack overflow posts using machine learning. *Concurrency and Computation: Practice and Experience* 33 (2021).

[2] Areti Ampatzoglou, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Paris Avgeriou, Angeliki-Agathi Tsintzira, and Alexander Chatzigeorgiou. 2021. Architectural decision-making as a financial investment: An industrial case study. *Information and Software Technology* 129 (2021), 106412. https://doi.org/10.1016/j.infsof.2020.106412

[3] Muhammad Ali Babar and Ian Gorton. 2007. A Tool for Managing Software Architecture Knowledge. In *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*. 11–11. https://doi.org/10.1109/SHARK-ADI.2007.1

[4] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. 2017. Automatic Extraction of Design Decisions from Issue Management Systems: A Machine Learning Based Approach. In *ECSA*.

[5] Manoj Bhat, Christof Tinnes, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. 2019. ADeX: A Tool for Automatic Curation of Design Decision Knowledge for Architectural Decision Recommendations. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. 158–161. https://doi.org/10.1109/ICSA-C.2019.00035

[6] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder. *CoRR* abs/1803.11175 (2018). arXiv:1803.11175 http://arxiv.org/abs/1803.11175

[7] Juliet Corbin and Anselm L. Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology* 13 (1990), 3–20. Issue 1.

[8] Klaas Andries de Graaf, Peng Liang, Antony Tang, and Hans van Vliet. 2015. Supporting Architecture Documentation: A Comparison of Two Ontologies for Knowledge Retrieval. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE '15)*. Association for Computing Machinery, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/2745802.2745804

[9] Barney G. Glaser and Anselm L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. de Gruyter, New York, NY.

[10] Neil B. Harrison, Paris Avgeriou, and Uwe Zdun. 2007. Using Patterns to Capture Architectural Decisions. *IEEE Software* 24, 4 (2007), 38–45. https://doi.org/10.1109/MS.2007.124

[11] Carsten Hentrich, Uwe Zdun, Vlatka Hlupic, and Fefie Dotsika. 2015. An Approach for Pattern Mining through Grounded Theory Techniques and Its Applications to Process-Driven SOA Patterns. In *Proceedings of the 18th European Conference on Pattern Languages of Program (EuroPLoP '13)*. Association for Computing Machinery, New York, NY, USA, Article 9, 16 pages. https://doi.org/10.1145/2739011.2739020

[12] Gregor Hohpe. 2006. Workshop Report: Conversation Patterns. In *The Role of Business Processes in Service Oriented Architectures (Dagstuhl Seminar Proceedings)*, Frank Leymann, Wolfgang Reisig, Satish R. Thatte, and Wil M. P. van der Aalst (Eds.), Vol. 06291. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[13] Matthew Honnibal and Ines Johnson. 2015. spaCy: Industrial-strength NLP with Python and Cython. https://github.com/explosion/spaCy.

[14] ISO/IEC/IEEE. [n. d.]. ISO/IEC/IEEE 42010 Standard. https://www.iso.org/standard/50508.html.

[15] A. Jansen and J. Bosch. 2005. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. 109–120. https://doi.org/10.1109/WICSA.2005.61

[16] R Burke Johnson and Larry Christensen. 2019. *Educational research: Quantitative, qualitative, and mixed approaches*. SAGE Publications, Incorporated.

[17] Staffs Keele et al. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report. Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

[18] Oliver Kopp, Anita Armbruster, and Olaf Zimmermann. 2018. Markdown Architectural Decision Records: Format and Tool Support. In *ZEUS*.

[19] Philippe B Kruchten. 2004. An Ontology of Architectural Design Decisions in Software-Intensive Systems. In *2nd Groningen Workshop on Software Variability*. Groningen, the Netherlands.

[20] Italo Carlo Lopes Silva, Patrick H. S. Brito, Baldoino F. dos S. Neto, Evandro Costa, and Andre Almeida Silva. 2015. A Decision-Making Tool to Support Architectural Designs Based on Quality Attributes. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*. Association for Computing Machinery, New York, NY, USA, 1457–1463. https://doi.org/10.1145/2695664.2695928

[21] Ioanna Lytra, Gerhard Engelbrecht, Daniel Schall, and Uwe Zdun. 2015. Reusable Architectural Decision Models for Quality-Driven Decision Support: A Case Study from a Smart Cities Software Ecosystem. In *2015 IEEE/ACM 3rd International Workshop on Software Engineering for Systems-of-Systems*. 37–43. https://doi.org/10.1109/SESoS.2015.14

[22] Ioanna Lytra, Patrick Gaubatz, and Uwe Zdun. 2015. Two Controlled Experiments on Model-Based Architectural Decision Making. *Inf. Softw. Technol.* 63, C (jul 2015), 58–75. https://doi.org/10.1016/j.infsof.2015.03.006

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[24] Henry Muccini and Karthik Vaidhyanathan. 2019. ArchLearner: Leveraging Machine-Learning Techniques for Proactive Architectural Adaptation. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2 (ECSA '19)*. Association for Computing Machinery, New York, NY, USA, 38–41. https://doi.org/10.1145/3344948.3344962

[25] Evangelos Ntentos, Uwe Zdun, Konstantinos Plakidas, Daniel Schall, Fei Li, and Sebastian Meixner. 2019. Supporting Architectural Decision Making on Data Management in Microservice Architectures. In *13th European Conference on Software Architecture (ECSA) - 2019*. http://eprints.cs.univie.ac.at/6071/

[26] Michael Nygard. 2011. Documenting Architecture Decisions. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions.

[27] Mahesh Parmar, W.U. Khan, and Binod Kumar. 2011. An Architectural Decision Tool Based on Scenarios and Non-functional Requirements. *International Journal of Advanced Computer Science and Applications* 2, 2 (2011). https://doi.org/10.14569/IJACSA.2011.020217

[28] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology* 64 (2015), 1–18.

[29] Chris Richardson. 2017. A pattern language for microservices. http://microservices.io/patterns/index.html.

[30] Dirk Riehle, Nikolay Harutyunyan, and Ann Barcomb. 2021. Pattern Discovery and Validation Using Scientific Research Methods. arXiv:cs.AI/2107.06065

[31] Mohamed Soliman, Paris Avgeriou, and Yikun Li. 2021. Architectural design decisions that incur technical debt — An industrial case study. *Information and Software Technology* 139 (2021), 106669. https://doi.org/10.1016/j.infsof.2021.106669

[32] Mohamed Soliman, Marion Wiese, Yikun Li, Matthias Riebisch, and Paris Avgeriou. 2021. Exploring Web Search Engines to Find Architectural Knowledge. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. 162–172. https://doi.org/10.1109/ICSA51549.2021.00023

[33] Martin Tamke, Paul Nicholas, and Mateusz Zwierzycki. 2018. Machine learning for architectural design: Practices and infrastructure. *International Journal of Architectural Computing* 16, 2 (2018), 123–143. https://doi.org/10.1177/1478077118778580 arXiv:https://doi.org/10.1177/1478077118778580

[34] Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla, and Muhammad Ali Babar. 2010. A comparative study of architecture knowledge management tools. *Journal of Systems and Software* 83, 3 (March 2010), 352–370. https://doi.org/10.1016/j.jss.2009.08.032 Relation: http://www.rug.nl/informatica/onderzoek/bernoulli Rights: University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science.

[35] Dan Tofan and Matthias Galster. 2014. Capturing and Making Architectural Decisions: An Open Source Online Tool. In *Proceedings of the 2014 European Conference on Software Architecture Workshops (ECSAW '14)*. Association for Computing Machinery, New York, NY, USA, Article 33, 4 pages. https://doi.org/10.1145/2642803.2642836

[36] Jeff Tyree and Art Akerman. 2005. Architecture decisions: demystifying architecture. *IEEE Software* 22 (2005), 19–27.

[37] Claes Wohlin, Per Runeson, Martin Hoest, Magnus C. Ohlsson, Bjoern Regnell, and Anders Wesslen. 2012. *Experimentation in Software Engineering*. Springer.

[38] Alla Zakurdaeva, Michael Weiss, and Steven Muegge. 2020. *Detecting Architectural Integrity Violation Patterns Using Machine Learning*. Association for Computing Machinery, New York, NY, USA, 1480–1487. https://doi.org/10.1145/3341105.3374008

[39] Shengnan Zhang, Yan Hu, and Guangrong Bian. 2017. Research on string similarity algorithm based on Levenshtein Distance. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE, 2247–2251.

[40] Olaf Zimmermann. 2011. Architectural Decisions as Reusable Design Assets. *IEEE Software* 28, 1 (2011), 64–69. https://doi.org/10.1109/MS.2011.3

[41] Olaf Zimmermann, Uwe Zdun, Thomas Gschwind, et al. 2008. Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method. In *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. IEEE, 157–166.