ADeX: A Tool for Automatic Curation of Design Decision Knowledge for Architectural Decision Recommendations

Manoj Bhat*, Christof Tinnes*, Klym Shumaiev*, Andreas Biesdorf†, Uwe Hohenstein† and Florian Matthes*
* Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
{manoj.mahabaleshwar,christof.tinnes,klym.shumaiev,matthes}@tum.de
†Siemens AG - Corporate Technology, Otto-Hahn-Ring 6, München 81739, Germany
{andreas.biesdorf,uwe.hohenstein}@siemens.com

***Abstract*—Software architecture can be seen as a set of architectural design decisions. These decisions, especially their rationale, play an important role in understanding software systems and constraints that are imposed on future design decisions. Documenting and managing these design decisions takes a lot of effort and is therefore often neglected by software architects. Following our vision to avoid explicit architectural knowledge management, we present an innovative tool used in industrial practise that is able to automatically recover design decisions from natural language text. The identified design decisions are then used to automatically derive further architectural knowledge, including related quality attributes and architectural elements to inhibit architectural erosion. By automatically enriching architectural knowledge and providing recommendations, the tool offers effective support for software architects in the decision-making process. The knowledge base of the tool is automatically kept up to date with information from different systems used by architects during the software development lifecycle process.**

***Keywords*-Architectural design decision; Architectural decision support; Architecture Knowledge Management; Tool; Recommendation;**

I. Introduction

The state of the art in software architecture is to consider software architecture not only as the overall structure of the software but as a set of architectural design decisions (ADDs) [1]. These ADDs capture the rationale of the decision-making outcome. Since ADDs introduce design rules and constraints on software architecture, it is essential that these ADDs, including their rationale, are available and accessible to software architects to make informed future ADDs.

The importance of documenting software architecture has often been stressed in the literature [2]. There are already tools for the management and retrieval of architectural knowledge (AK). For example, semantic wiki-based tools like ArchiMind [3] and ArchiMedes [4] not only enable architects to store AK but also enable them to maintain the metadata and relationships. With tools such as Archium [1], the architecture can be constructed as a set of ADDs, and the tool can be used to capture alternatives for design decisions. A detailed comparison of a plethora of AK management (AKM) tools is given in [5] and [6].

Unfortunately, even though, architects understand the benefits of documenting design decisions, in practice, those *decisions are rarely documented* [7], and they are lost over time. In an attempt to resolve this dilemma, two apparent approaches are typically used in industrial practice: (a) either setting incentives for architects to improve their willingness to document their decisions or (b) automatically capturing or generating architectural documentation. In fact, AK is already implicitly captured in artifacts that are the results of the day-to-day activities of software architects and developers, e.g., issues in Issue Management Systems (IMS), meeting minutes, chat histories, and source code commits. Based on the learnings from the existing AKM tools, the future research directions proposed in literature (e.g., from [8]), and in collaboration with our industry partner (a software architecture definition department within a large multinational company), we have developed "ADeX" (**A**melie - **De**cision E**x**plorer) over the last four years. We target the bottom-up approach to AKM with the focus on automating the AK curation process. We agree with Tang et al. regarding the fact that ADDs are not explicitly recorded [9] and the existing AKM tools are considered intrusive since they require architects to manually document ADDs which can be tedious and expensive. ADeX, therefore, aims to automatically extract, enrich, and generate specific views on AK to support architects' decision-making process. The tool is based on a conceptual framework described in detail in [10].

II. Use cases for a Decision Explorer

ADeX is designed to be used by architects and developers, but project managers can also profit from the information provided by this tool. ADeX supports the following main use cases (which have been derived from [11]):

UC1: To quickly get insights into a project, users can obtain an overview of design decisions including their classification according to the decision types, affected quality attributes, and architectural elements (AEs).

UC2: Users can obtain an overview of AEs used in the project and their relevance. This use case could be used for insights into the project or staffing purposes (since it shows applied technologies and concepts).

UC3: To see the dynamics and evolution of a project, users can perform retrospective analyses of design decisions,

quality concerns, and AEs, i.e., the overview from UC1 and UC2 are aggregated year-wise. This use case gives insights into which quality concerns and AEs are becoming less or more relevant.
UC4: To assist users in choosing technologies to realize a design decision, they get suggestions about possible software solutions for a selected design decision.
UC5: To assist users during design space exploration, alternative solutions for a design decision are recommended.
UC6: Users get suggestions about experts who can be involved in solving new design concerns.
UC7: For each decision, users obtain a list of similar decisions made in the past. It supports the reuse of design decisions.

We have developed ADeX mainly for three purposes: Firstly, ADeX is used for getting insights especially into large software-intensive projects. Architects and developers can use ADeX to browse through projects and get a quick overview of the already made design decisions. These design decisions are automatically identified, for instance, from issues in an IMS. Further related information regarding those design decisions can be explored, such as the affected quality attributes or AEs (concepts like databases or concrete technologies like SQL). An overview of the concepts and technologies used in a project could also be used by project managers for staffing purposes. Secondly, ADeX is used to support architects and developers during the decision-making process. For instance, unresolved design concerns from an IMS are extracted into ADeX's knowledge base, and after preprocessing, architects get recommendations for resolving those concerns. ADeX helps users answer questions such as "who should be involved in decision making?" and "which similar decisions have been made in the past?". Furthermore, architects instead of exploring the solution space, often choose solutions (e.g., software products) based on their intuition and past experiences (the so-called availability bias [12]). We, therefore, integrated an automatic recommendation mechanism that shows software solutions and alternatives related to AEs in a given design decision. This triggers the mental process of architects and helps them to reflect on their choices.

III. Architecture

The architecture of ADeX is based on the conceptual framework for architectural decision making, which has been described in [10]. ADeX is a collaborative web application. The user interfaces (UIs) are designed according to Google's Material Design[1]. The back-end of ADeX follows a microservice architecture. For persistence, the MongoDB[2] and a meta-model based system called SocioCortex [13] are used. The domain model of AK maintained within SocioCortex can be adapted at runtime to fit the needs of different software engineering projects. SocioCortex provides REST-APIs which are used by other ADeX components as shown in Figure 1.

[1]https://material.io/design/
[2]https://www.mongodb.com/

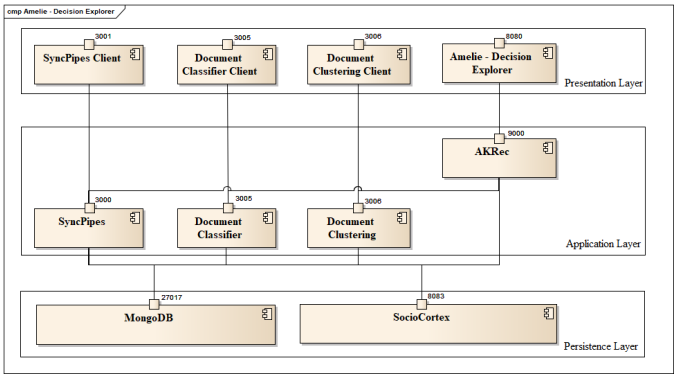

Fig. 1. High-level architecture of ADeX

A. SyncPipes

SyncPipes is a model-based extract-transform-load (ETL) component that allows integrating various data sources, transforms the imported data to align with a target data model, and loads the transformed data into the target system. The source and target data models are described using JSON-Schema. SyncPipes uses a handler, which starts a synchronization job that triggers the ETL process based on specific events or regular time intervals to keep the sources and the system synchronized. For ADeX, the SyncPipes's target system is the AKM system (SocioCortex), and the source systems include JIRA, Github issues, MS Project, and Enterprise Architect. The pipelines within SyncPipes can be configured via a front-end web application, and the configurations are stored in a MongoDB database.

B. Document Classifier

Document Classifier is used to classify textual data (e.g., extracted from JIRA via SyncPipes) as design decisions using supervised machine learning (ML). The classification is done in a two-phase process (cf. [14] for details). In the first phase, a binary classifier automatically labels a given issue as either a "design decision" or "not a design decision". In the second phase, design decisions are further classified into one of the three decision categories (structural, behavioral, and non-existence/ban decision). Using the Weka library, summary and description of issues are preprocessed, and the vector representations of the issues are generated. These vector representations are fed into a Support Vector Machine (SVM) classifier which labels those corresponding issues. The Document Classifier component is implemented using a Java-based web application framework[3]. Information related to the classification pipelines is stored in MongoDB, and the classification models are persisted on the file system. The classification of issues as design decisions is a prerequisite for other functionalities that follow.

C. AKRec - Architectural Knowledge Recommender

The AKRec component and the corresponding front-end (see below) address the use cases described in the previous

[3]https://www.playframework.com/

section. It uses the identified design decisions within a project and enriches them with information such as affected quality attributes, AEs, alternative options, and individuals' expertise on specific AEs. Furthermore, this component serves as a middleware which invokes all the preprocessing steps, aggregates results, and provides them to the client application for presentation. AKRec uses the DBPedia ontology for identifying and annotating AEs within the textual description of design decisions. The annotators are based on the UIMA framework[4]. Using the DBPedia ontology, alternative choices for a given AE are determined by executing complex SPARQL queries (cf. [15] for details). Therefore, the AEs, alternatives, and software solutions don't need to be manually maintained but are automatically retrieved from external sources. Annotations for each design decision and their recommendations are also stored in the MongoDB. Persisting this information in the database improves the performance of the application since for already annotated design decisions the expensive annotation and querying steps can be omitted.

To capture the rationale behind design decisions, quality attributes (from ISO/IEC 25010) are automatically mapped to the identified design decisions. To map the quality attributes to design decisions, we use a string-matching algorithm to find synonyms and keywords for each the quality attribute in the textual description of design decisions. The mappings are persisted in the MongoDB as well.

In order to recommend experts who can be involved in decision making, architects' and developers' expertise about AEs are quantified using an expertise matrix. The AEs contained in the design decisions are the columns of this matrix and rows represent architects and developers. Each matrix entry is then computed as the number of design decisions which contain the corresponding AE and are resolved by the corresponding architect or developer. The rows represent the expertise of a single architect and are called expertise profile (EP). To make expert recommendations for new design concerns, a frequency vector, called concept vector (CV), of the AEs within the textual description of the concern is computed. The score for each expert is then computed as $score = \|EP \circ CV\|$, where "$\circ$" is the entry-wise multiplication of the two vectors. It should be noted that the expertise matrix can not only be used to identify people with expertise but also helps to distribute knowledge among team members and therefore prevent the formation of knowledge islands or hotspots. The expertise matrix for each project is persisted in the MongoDB. See [16] for a detailed description of the recommendation algorithm.

D. Document Clustering

The Document Clustering component creates and persists a document cluster model of all the identified design decisions using an unsupervised machine learning algorithm. These clusters of similar design decisions are created using the Word2Vec representation of design decisions' description and K-means clustering algorithm thereafter. A new design concern is compared with the persisted cluster model to identify similar design decisions made in the past[5]. Those identified similar decisions are ranked according to their similarity score and presented to the users as described below.

E. AMELIE - Decision Explorer client

This component is the main front-end of ADeX. The AKRec component exposes its annotation and recommendation services as REST-APIs to this client. Figure 2 shows one viewpoint (related to quality attributes) within this component. The front-end is built using the node.js[6] environment and relies on React[7] for its UI components. For graphical visualization, the D3.js[8] library is used. The front-end features the following views: First, in the *projects overview page*, users can browse and search through a list of projects (which is again automatically extracted using the SyncPipes component). From the *project import view*, users can trigger the import of JIRA issues (or other project information such as tasks from MS Project or Github, requirements from Enterprise Architect or Excel files). Furthermore, the decision classification, clustering, annotation, and computation of the expert matrix is automatically triggered by the import operation. Second, in the *quality attributes view*, as shown in Figure 2, an overview of design decisions is given including their classification and affected quality attributes. Third, the *AEs view* presents a bubble chart diagram of the identified AEs; the bubble size is proportional to the frequency of the AEs affected by design decisions. The quality attributes view and the AEs view incorporate a time slider feature, where users can browse the affected quality concerns or AEs year-wise. Fourth, the *expert matrix view* depicts the expertise of architects and developers for the AEs. Using an integrated search, users can find individuals who have expertise in specific topics. Fifth, in the *expert recommender view*, a list of architects and developers sorted according to their expertise score is recommended to address open design concerns. Sixth, in the *design decisions view*, users can browse through all the design decisions identified within a project. This view shows the title, description, quality attributes, architectural elements, decision categories, and the status of each design decision. Clicking on a design decision leads to the final *annotation and alternatives view*. In this view, the description of the selected design decision is annotated with AEs and selecting an AE lists the possible alternatives and software solutions. Users can also add, update, or remove these recommendations. For the selected design decision, another tab below the recommendations view shows a list the similar design decisions along with the similarity scores. The online version of the tool is available here: https://amelietor-9f8c3.firebaseapp.com

[4]https://uima.apache.org/

[5]See the Master thesis for a detailed description about the approach and the algorithm parameters: https://wwwmatthes.in.tum.de/pages/l2inbfu3sbe7

[6]https://nodejs.org/en

[7]https://reactjs.org/

[8]https://d3js.org

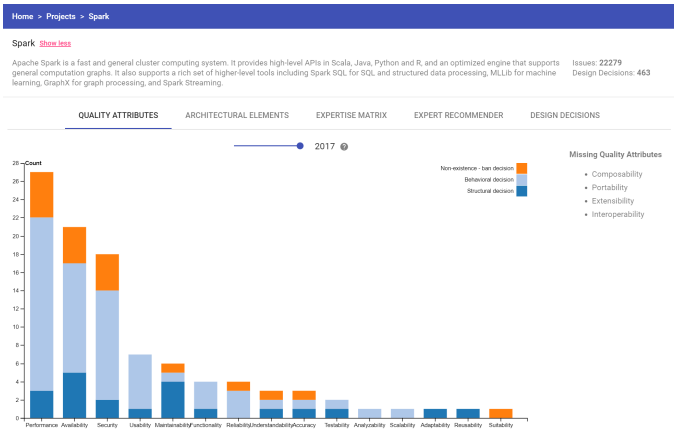

Fig. 2. The quality attributes view on design decisions (x-axis represents quality Attributes, y-axis indicates the count of related design decisions)

IV. Evaluation, Conclusion and Outlook

In this paper, we have presented the tool named ADeX for the automatic curation of design decisions and supporting architects and developers during the decision-making process. The tool follows our idea that forcing software architects to document their design decisions is often unreasonable and therefore, bottom-up approaches to software architecture documentation and AK curation should be the way forward for AKM in industrial settings. Various components within ADeX have been developed over the last four years in collaboration with and our industry partner who provided practical feedback. Hence, the components have been evaluated both qualitatively and quantitatively. The quantitative evaluation of the components has been discussed in their respective publications referred earlier. For instance, we have shown in [14] that design decisions can be automatically extracted from issues with an accuracy (F-score) of 91.29%, AEs can be automatically annotated with an accuracy of 84%, and with an accuracy of 79.64% experts can be recommended to address specific design concerns.

Regarding the qualitative evaluation, we received feedback comprising of areas of improvement, positive, as well as, negative aspects of ADeX from architects and developers in five different industrial projects. For instance, concerning the AEs view, architects suggested that heightening the relationships between AEs in the bubble chart would benefit the impact analysis of decisions on related AEs. Similarly, even though we present related similar decisions made in the past, it was suggested that identifying the type of relationships among similar decisions would be beneficial. We are currently working on this suggestion to automatically extract relations (such as *related to*, *decomposes into*, *constraints*, and *contradicts*) from the data sources. Finally, with regards to the expertise matrix, even though the benefits of such a matrix with the integrated search was mentioned, at least three architects commented the following: "some people [architects and developers] might not be happy that you measure their expertise". We do realize that special care must be taken and workers' council needs to be involved in setting up and using such a system in industry.

For future work, we plan to integrate some of the functionality into the commonly used design, development, and project management software and processes. Furthermore, since AK reuse reduces the design costs and also improves the quality, another important step for our future work is to investigate how to transfer and reuse AK between different projects.

References

[1] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *WICSA 2005*. IEEE, 2005, pp. 109–120.

[2] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures Views and Beyond*. Addison-Wesley Professional, 2003.

[3] K. A. de Graaf, "Annotating software documentation in semantic wikis," in *Proceedings of the fourth workshop on Exploiting semantic annotations in information retrieval*. ACM, 2011, pp. 5–6.

[4] R. C. de Boer, "Archimedes publication and integration of architectural knowledge," in *ICSAW 2017*. IEEE, 2017, pp. 268–271.

[5] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar, "A comparative study of architecture knowledge management tools," *J. Syst. Softw.*, vol. 83, no. 3, pp. 352–370, 2010.

[6] M. Shahin, P. L. P. Liang, and M. Khayyambashi, "Architectural design decision: Existing models and tools," *2009 Joint WICSA & ECSA*, no. March, pp. 293–296, 2009.

[7] J. F. Hoorn, R. Farenhorst, P. Lago, and H. Van Vliet, "The lonesome architect," *J. Syst. Softw.*, vol. 84, no. 9, pp. 1424–1435, 2011.

[8] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *J. Syst. Softw.*, vol. 116, pp. 191–205, 2016.

[9] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *J. Syst. Softw.*, vol. 79, no. 12, pp. 1792–1804, 2006.

[10] M. Bhat, K. Shumaiev, and F. Matthes, "Towards a framework for managing architectural design decisions," in *Proc. 11th ECSA: Companion Proc.* ACM, 2017, pp. 48–51.

[11] J. Xu, "Improving the usability of an integrated decision support system for design decision making," Master's thesis, Technical University of Munich, Germany, 2018. [Online]. Available: https://wwwmatthes.in.tum.de/pages/1r15fyryyv3gc

[12] A. Manjunath, M. Bhat, K. Shumaiev, A. Biesdorf, and F. Matthes, "Decision making and cognitive biases in designing software architectures," in *ICSA-C 2018*. IEEE, 2018, pp. 52–55.

[13] T. Reschenhofer, M. Bhat, A. Hernandez-Mendez, and F. Matthes, "Lessons learned in aligning data and model evolution in collaborative information systems," in *IEEE/ACM ICSE-C*. IEEE, 2016, pp. 132–141.

[14] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, "Automatic extraction of design decisions from issue management systems: a machine learning based approach," in *ECSA 2017*. Springer, 2017, pp. 138–154.

[15] M. Bhat, K. Shumaiev, A. Biesdorf, and F. Matthes, "An ontology-based approach for software architecture recommendations," in *23rd AMCIS 2017, Boston, MA, USA, August 10-12, 2017*, 2017.

[16] M. Bhat, K. Shumaiev, K. Koch, U. Hohenstein, A. Biesdorf, and F. Matthes, "An expert recommendation system for design decision making: Who should be involved in making a design decision?" in *ICSA 2018*. IEEE, 2018, pp. 85–8509.