Automatic Extraction of Design Decisions from Issue Management Systems: A Machine Learning Based Approach

Manoj Bhat*, Klym Shumaiev*, Andreas Biesdorf+, Uwe Hohenstein+, and Florian Matthes*

*Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
{manoj.mahabaleshwar,klym.shumaiev,matthes}@tum.de
+Siemens AG - Corporate Technology, Otto-Hahn-Ring 6, München 81739, Germany
{andreas.biesdorf,uwe.hohenstein}@siemens.com

Abstract. The need to explicitly document design decisions has been emphasized both in research and in industry. To address design concerns, software architects and developers implicitly capture design decisions in tools such as issue management systems. These design decisions are not explicitly labeled and are not integrated with the architecture knowledge management tools. Automatically extracting design decisions will aid architectural knowledge management tools to learn from the past decisions and to guide architects while making decisions in similar context. In this paper, we propose a two-phase supervised machine learning based approach to first, automatically detect design decisions from issues and second, to automatically classify the identified design decisions into different decision categories. We have manually analyzed and labeled more than 1,500 issues from two large open source repositories and have used this dataset for generating the machine learning models. We have made the dataset publicly available that will serve as a starting point for researchers to further reference and investigate the design decision detection and classification problem. Our evaluation shows that by using linear support vector machines, we can detect design decisions with 91.29% accuracy and classify them with an accuracy of 82.79%. This provides a quantitative basis for learning from past design decisions to support stakeholders in making better and informed design decisions.

Keywords: Software architecture, Design decisions, Machine learning

1 Introduction

Over the last decade, there has been a paradigm shift in how we view software architectures. Since the representation of Architectural Design Decisions (ADDs) as first-class entities [5, 17, 32], software architecture is considered as a set of architectural design and ADDs [15, 18]. The architectural knowledge management (AKM) tools [3, 4, 21, 23] support the documentation of ADDs and its associated concepts including architectural concerns, alternative architectural

solutions, and rationales for ADDs. Moreover, industry standard software architecture templates (for example, arch42[1]) provide placeholders to capture ADDs. Documenting ADDs supports stakeholders to understand and reason about the software architecture during both the development and maintenance phases [6]. However, the manual effort [8, 19], time, and cost [31] involved in the documentation process are a concern for practitioners and its immediate benefit is not visible [20]. Hence, industry has often not recognized the value of ADDs, for example, by taking benefit from reoccurring design concerns in similar context.

Furthermore, with the rapid adoption of agile methodologies for software development, ADDs both in large open-source software (OSS) and in industrial projects are scarcely documented [1]. However, stakeholders involved in projects, that follow this agile movement, tend to use agile project management tools such as issue trackers and version control systems [29, 30]. In such projects, even though design decisions are not explicitly documented, they are implicitly captured in different systems including project management, issue management, source code version management, and meeting recording systems [25].

The use of issue management systems (for example, JIRA and GitHub issue tracker) for managing issues is becoming popular both in industrial settings as well as in OSS projects [2, 12]. An issue is either a task, new feature, user story, or bug. These systems provide a common interface for stakeholders to track, communicate, and visualize the progress of tasks within a project. For instance, a software architect can create a new task (which might implicitly represent a design decision) such as "Remove dependency on Twitter4J repository[2]" and assign it to a developer to complete the task. Furthermore, as a good practice, the developer community maintains a link between the task in the issue management system and the source code commits or pull-requests in version control systems using the task identifier or vice versa. In sum, issue management systems are an excellent source that implicitly captures decisions made by architects and developers [25] and acts as a bridge between stakeholders' requirements and the source code of the corresponding software system. Furthermore, the attributes such as reporter, assignee, and creation date of the issue are also maintained in such systems and can be used to enrich the meta-information of design decisions in AKM tools, for example, to refer to originators and experts.

In this paper, we address the extraction and classification of design decisions that are not systematically documented in AKM tools but implicitly captured in issue management systems. The contribution of this paper is twofold. First, we propose a two-phase machine learning (ML) based approach (cf. Figure 1). In the first phase, design decisions are automatically detected from issues that are extracted from an issue management system. In the second phase, the identified design decisions are further classified into different decision categories. Second, we make the manually labeled dataset, which was created for training the ML models publicly available[3]. Since, no such labeled dataset exists, this contribution

[1] `http://arc42.org/`
[2] `https://issues.apache.org/jira/browse/SPARK-710`
[3] `https://server.sociocortex.com/typeDefinitions/1vk4hqzziw3jp/Task`

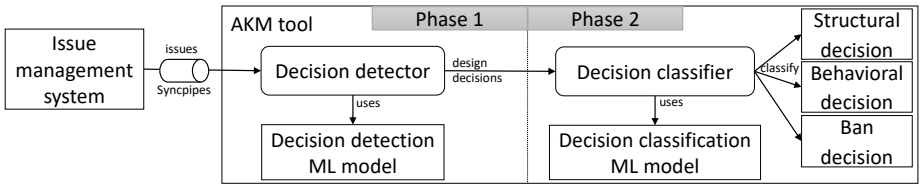

Fig. 1. A two-phase ML-based approach for decision detection and classification

will serve as a starting point and reference for researchers to apply and compare supervised ML algorithms for extracting and classifying design decisions.

As discussed by Kazman et al. [16], architecture needs to be made explicit to avoid knowledge vaporization and to favor the knowledge acquisition process for newcomers and adopters of the project. The extraction of design decisions from issues will support the process of capturing AK in the AKM tools, which will in turn enable various use cases including reasoning, recommendations, traceability, and report generation for stakeholders. In addition, automatically classifying the extracted decisions into different categories such as Structural, Behavioral, and Ban decisions (cf. Section 2) will label those decisions to aid the search and the recommendation use cases. In particular, it will allow the creation of a knowledge base that can be used, for instance, to learn from the decisions made in similar past projects. Software architects will be able to rely on decisions made in the past to address design concerns in their current projects. As van der Ven and Bosch [34] put it, "Wouldn't it be great if software architects could get access to the decisions made by other architects, that would allow them to determine what selections were made from a set of alternatives and with what frequency?"

This paper is organized as follows. Section 2 describes the related work. In Section 2, we revisit the ADD categories proposed by Kruchten. Section 4 presents the dataset preparation process. Section 5 describes the setup of the ML pipeline used for decision detection and classification. The results of applying different multi-class classification algorithms are discussed in Section 6. Finally, we conclude with a short summary and an outlook on the future research.

2 Related Work

The need to systematically capture design decisions to enable reasoning and decision support in AKM tools has been extensively discussed in the past. For instance, Babar and Gorton [3] propose an AKM tool named PAKME for managing architectural knowledge and rationale. The repository within PAKME consists of generic design options and architectural patterns that can be assessed by architects before making architectural decisions. Similarly, tools such as Decision Architect [23] and ADvISE [21] allow architects to capture and analyze architectural decisions. Capilla et al. [7] in their literature study analyze these tools and their functionalities and indicate that there is a need for substantial improvement in the ability to (semi-) automate use cases for AKM.

The aforementioned tools follow a top-down approach to AKM, that is, they require stakeholders to manually capture data in respective tools which then enables traceability and reasoning based on their meta-models. However, architectural documentation is sparse and stakeholders tend to rather use agile tools such as issue trackers, e-mail clients, PowerPoint, and meeting recording systems to capture their day-to-day decisions [25]. As compared to the top-down approach, we envision a bottom-up approach that focuses on analyzing existing data to automatically extract design decisions and structure them thereafter.

The research in the area of automatic design decision detection and classification is still in its infancy. The approach taken by van der Ven and Bosch [34] is closely related to our work. They propose an approach for analyzing design decisions maintained in the source code commits of OSS repositories. In their work, six subject matter experts manually analyzed 100 different commit messages and indicated that 67% of those commit messages reflected design decisions. Similarly, based on surveys, Dagenais and Robillard [10] identified decisions from developer documentation. In our work, however, we study the issues maintained in issue management systems and apply a ML-based approach to automatically extract and classify design decisions.

Furthermore, in [11] and [13], authors have successfully applied speech analysis techniques to automatically detect decision-related conversations. We believe that such efforts to automatically detect and extract decisions from systems that are extensively used by architects and developers will aid the adoption of AKM tools to provide significant decision support. Hence, in this paper, we focus on extracting and classifying design decisions from one of the frequently used systems in software development, that is, issue management systems.

3 ADD Categories

In his seminal work [17], Kruchten introduced an ontology of ADDs in software-intensive systems. He classified ADDs into three main categories – existence decisions, property decisions, and executive decisions. Figure 2 shows the taxonomy of the ADD categories with the emphasis on existence decisions, which is the focus of our proposed approach.

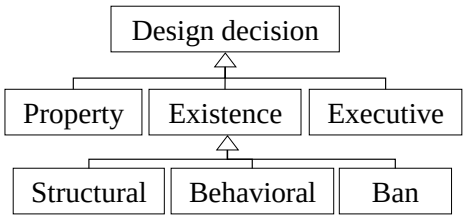

Fig. 2. ADD categories (source: [17])

Existence decisions: Decisions that reflect the existence of an artifact in a system's design or implementation. These decisions are further classified into ***structural***, ***behavioral***, and ***ban*** or ***non-existence*** decisions. Those decisions that indicate the creation or update of artifacts in a system are referred to as structural decisions. Whereas, those decisions that capture, for instance how components interact with each other or discuss the functionality of the system

are referred to as behavioral decisions. For example, "Add jets3t dependency to Spark Build[4]" corresponds to a structural decision and the task "Add job cancellation to PySpark[5]" is a behavioral decision. Finally, those decisions that result in the removal of an artifact or interaction between artifacts are referred to as ban or non-existence decisions. For example, the task "remove numpy from RDDSampler of PySpark[6]" is a ban decision. As discussed in [17,25], identifying and then documenting ban decisions is important since these decisions are not traceable to any existing system artifacts.

Property decisions influence the general quality of a system. Design rules, guidelines, and design constraints are considered as property decisions.

Executive decisions are driven by the business environment, management processes, and organizational structures.

Miesbauer and Weinreich [25] demonstrated in their expert survey that the majority of design decisions are existence decisions. In total, they collected 120 examples of design decisions during the interviews. After mapping the examples to the decision categories, they noted that 65% of decisions were existence decisions, 27% were executive decisions (most of them technology decisions), and the remaining 8% belonged to the property decision category. With this as a basis, as well as due to the high manual effort involved in the labeling process for generating the dataset, we start our analysis by considering existence decisions along with its three sub-categories. However, by creating labeled data for the remaining categories and then by training the supervised classifiers, the proposed approach can be extended.

Apart from the aforementioned categories, it should also be noted that design decisions could also be classified according to different abstraction levels. Jansen [14] proposes a funnel of decision-making model to classify decisions at different abstraction levels such as software architecture, detailed design, and implementation. Van der Ven and Bosch [34] relate to these abstraction levels as *high-level*, *medium-level*, and *realization-level* decisions. The decisions at different abstraction levels are related to each other and form a tree structure. Moreover, the decisions at a higher level of abstraction constraint or influence the decisions at lower levels. We observed during the manual analysis that the decisions extracted from issues belong to either medium-level or realization-level decisions. Software architects and developers make these decisions during the implementation and maintenance phase of a project. Moreover, since these decisions are the hardest to make [33, 34], extracting and recommending them to software architects will support the decision-making process in similar projects. In order to achieve this, we first need to identify, extract, and classify design decisions from the existing projects. Hence, we formulate the following two hypothesis:

1. Design decisions can be automatically identified and extracted from issues.
2. Design decisions can be automatically classified into ADD categories, namely structural, behavioral, and ban decisions.

[4] `https://issues.apache.org/jira/browse/SPARK-898`
[5] `https://issues.apache.org/jira/browse/SPARK-986`
[6] `https://issues.apache.org/jira/browse/SPARK-4477`

To validate the aforementioned hypothesis, we used issues maintained in two large OSS projects. We first extracted the issues from an issue management system into an AKM tool. Then, we applied a ML-based approach to a) automatically extract design decisions from the already extracted issues and b) automatically classify the extracted design decisions into three specific categories. The dataset preparation process and the ML pipeline setup for generating the ML models are elaborated in the subsequent sections.

4 Dataset

In this Section, we present the data extraction, curation, and manual labeling processes for generating the dataset for decision detection and classification.

4.1 Data Extraction

We considered two large OSS projects, namely *Apache Spark* and *Apache Hadoop Common* for this study. Apache Spark is a large-scale data processing engine. Since early 2014, contributors of this project have captured more than 19,000 publicly accessible issues in JIRA from version 0.9.0 to 2.1.0[7]. Apache Hadoop, on the other hand, is a distributed computing software and the Hadoop Common component is the core that provides utilities to the other Hadoop components such as YARN and MapReduce. Hadoop Common maintains more than 10,000 issues from version 0.2.0 to 3.0.0-apha1, since early 2013[8]. Both these projects are related to each other, as Apache Spark runs in Hadoop clusters. We selected these two projects for the following reasons:

- Interest to analyze design decisions for building a data analytics platform
- Experts responsible for generating the training dataset for ML had used either one of the systems and were involved in data analytics projects
- Both are long-running projects and have maintained more than 10,000 issues
- Both these projects are extensively used in data management solutions[9]

During the extraction process, we extracted the issues related to these two projects from JIRA while filtering for the following relevant settings. The list of prerequisites for issues to qualify for our study helped us to narrow down the large number of issues to those issues that potentially reflect design decisions. For instance, a *critical task* that has been *resolved* by *implementation* indicates that there is a potential change in the detailed design of a software system.

- Issue Type = Task, New Feature, Improvement, or Epic
- Priority = Blocker, Critical, or Major
- Status = Resolved
- Resolution = Fixed, Implemented, Done, or Resolved

[7] `https://issues.apache.org/jira/browse/SPARK` – last accessed on 25.01.2017
[8] `https://issues.apache.org/jira/browse/HADOOP` – last accessed on 25.01.2017
[9] `https://www.gartner.com/doc/3371732/critical-capabilities-data-warehouse-data`

To extract issues from JIRA, we used an OSS component within our AKM tool [4] named SyncPipes[10]. SyncPipes allows end users to map the properties of the source system (JIRA) to the target system (AKM tool). Subsequently, based on the properties mapping, a pipeline is established to enable data integration and synchronization. In total, we extracted 2,259 issues from Apache Spark and 420 issues from Hadoop Common projects.

4.2 Data Curation

We consider the **summary** and **description** attributes of an issue since they elaborately describe an issue's purpose. It should be noted that comments within issues could also be analyzed in this context. However, we restrict our data analysis to the text captured in summary and description attributes and consider the inclusion of comments as part of our future work.

As a first step, the summary and description of all the extracted issues were cleaned by removing the following:

- Code snippets within the text, as well as code inside {{ }} and {code} blocks
- Comments inside {noformat} blocks
- URLs inside the text

We introduced the above restriction so as to ensure that the intent of the issue can be justified only on the basis of textual description without the need for code snippets for explanation.

4.3 Manual Labeling

Two software architects with more than five years of experience individually analyzed the extracted issues in two steps. In the first step, these architects manually classified a set of issues into two classes, namely *Design Decision* and *Not A Design Decision*. In the second step, the decisions identified in the first phase were manually classified into three decision classes, namely *Structural decision*, *Behavioral decision*, and *Ban decision* (cf. Section 2). These steps were not necessarily carried out sequentially, but as per the convenience of the experts.

Before starting the labeling process, to ensure a common understanding between the two architects, we set up the rules presented in Table 1 for the manual classification. The classification of design decisions is purely based on the definition of decision categories as discussed in Section 2. To the best of authors' knowledge, there does not exist any design decisions dataset that can be used for reference. Hence, we put forth the rules shown in Table 1, for the two architects to support the manual labeling process.

Based on the aforementioned rules, both the architects manually analyzed the text in the summary and description attributes of all the extracted issues. Those issues with a missing description and whose intent was not explanatory using the textual description were marked as deleted. The issues that belonged

[10] `https://wwwmatthes.in.tum.de/pages/2gh0u9d1afap/SyncPipes`

Table 1. Rules for manual classification

Structural decision:

+ Adding or updating plugins, libraries, or third-party systems
+ Adding or updating classes, modules, or files (a class, in this context, refers to a Java class)
+ Changing access specifier of a class
+ Merging or splitting classes or modules
+ Moving parts of the code or the entire files from one location to another (code refactoring to address maintainability issues)
+ Updating names of classes, methods, or modules

Behavioral decision:

+ Adding or updating functionality (methods/functions) and process flows
+ Providing configuration options for managing the behavior of the system
+ Adding or updating application programming interfaces (APIs)
+ Adding or updating dependencies between methods
+ Deprecating or disabling specific functionality
+ Changing the access specifiers of methods

Ban decision:

+ Removing existing plugins, libraries, or third-party systems
+ Discarding classes, modules, code snippets, or files
+ Deleting methods, APIs, process flows, or dependencies between methods
+ Removing deprecated methods

Design decision:

+ An issue that belongs to any one of the above categories

Not a design decision:

+ An issue that does not belong to any of the above categories

to a specific decision category were labeled respectively, as well as, marked as a *Design Decision*. However, the issues that did not belong to any of the decision categories were marked as *Not A Design Decision*. During this process, we observed that some of the issues were abstract, in the sense that, they were broad issues that could be classified into more than one category. For example, the issue titled "Implement columnar in-memory representation[11]" aims to improve the memory efficiency of the system and represents a design decision. This issue affects the behavior of the system by introducing a new functionality and affects the structural aspects by introducing new Java classes for its implementation. In this study, we do not apply multi-label classification and focus only on multi-

[11] `https://issues.apache.org/jira/browse/SPARK-12785`

class classification[12] and hence, we restrict the labeling of issues to only one label. Moreover, the majority of issues could be classified into one category since issues are typically concise so that developers can easily understand and implement the tasks. To sum up, architects were requested to mark issues belonging to more than one category as deleted since we argued that applying multi-class classification for detection and classification of design decisions is sufficient to validate the hypothesis set for this study.

Once the architects labeled all the issues individually, the training dataset was consolidated with two focus points in a shared meeting.

- All those issues that were marked as deleted by both the architects were removed from the knowledge base.
- All those issues that had inconsistent decision categories were also removed. Since inconsistent dataset results in unreliable classification results, this step ensured that the issues in the dataset were labeled correctly.

The labeling process resulted in a dataset with 2,139 issues with **781** issues labeled as Design Decisions and 1,358 issues labeled as Not A Design Decision. To avoid skewed results towards Not A Design Decision label (due to a higher number of issues labeled as Not a Design Decision), we randomly selected **790** issues labeled as Not A Design Decision for generating the design decision detection ML model. Furthermore, out of 781 design decisions, 226 were labeled as Structural, 389 were labeled as Behavioral, and the remaining 166 as Ban design decision. To ensure a balanced input for generating the ML model for design decision classification, we randomly selected **160** issues from each category.

5 Machine Learning Pipeline

We used the pipeline shown in Figure 3 to generate the ML model for decision detection and decision classification. The pipeline itself was divided into two parts. In the first part – "process documents", the labeled dataset was the input and the pipeline generated the term frequency representation of issues. The output of the first part was then consumed by the second part – "Generate model" to produce the classification model and the result of applying the model on the testing dataset. Each issue in the labeled dataset was first tokenized to retrieve words. All the words were then transformed to lower cases. Stop words such as articles, conjunctions, and prepositions were removed. The remaining words were then stemmed to their root words using the Porter stemming algorithm [26]. Subsequently, a list of generated n-grams was appended to the word list. Generating n-grams helps to maintain the context of the usage of specific terms by considering its surrounding terms. For the evaluation, we tried different values of n (from 1 to 5) and documented the results as presented in the next section.

[12] Given that there are multiple labels, in multi-class classification, a document can be assigned to one and only one label. Whereas, in multi-label classification, a document can be assigned to any number of labels.

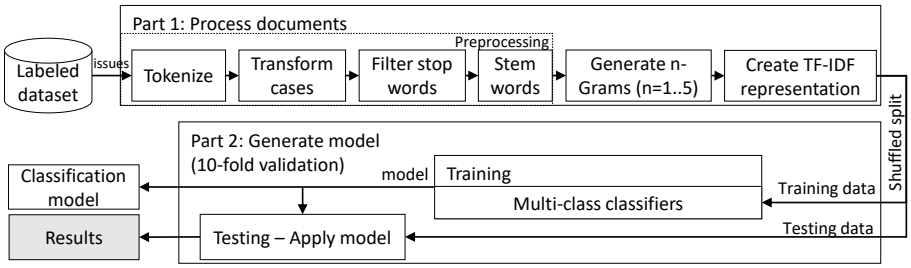

Fig. 3. The machine learning pipeline for design decision detection and classification; Classifiers: SVM, Naive Bayes, Decision tree, Logistic regression, One-vs-rest; n-grams: one to five; Split strategies: 90%, 80%, 70%, 60%, 50%;

Finally, the list of words was converted into a term frequency representation. For decision detection with a labeled dataset of 1,571 issues (781 and 790 issues labeled as design decision and not a design decision respectively), we used the term frequency-inverse document frequency (tf-idf) for vector representation. The tf-idf representation evaluates the number of times a word appears in an issue but is offset by the frequency of the word in the corpus. However, for decision classification, we only used term frequency since the dataset was comparatively smaller with 480 design decisions (160 issues in each decision category).

The term frequency representation of issues is provided as input to the second part of the pipeline for generating the classification model. We used different shuffled split strategies (90%, 80%, 70%, 60%, and 50%) and observed the results. That is, the documents were split into training dataset and testing dataset with different split percentages during multiple runs. Furthermore, we used k-fold cross-validation in the model generation process for estimating the accuracy. In our test runs, we used 10-fold cross-validation (k=10) which is common in data mining and machine-learning as it produces less biased accuracy estimations for datasets with small sample sizes [27]. We used different multi-class classifiers on the dataset with the parameters shown in Table 2. The classification model was then applied on the testing dataset to generate the classification results.

We implemented the pipeline shown in Figure 3 using Spark's scalable machine learning library (MLlib) [24]. The MLlib component provides interfaces to create and execute the pipe and filter based pipelines. The pipeline with its configurations and the generated model was eventually persisted as a Spark model instance in the AKM tool for subsequent decision classification. That is, for au-

Table 2. Classifier parameters

Support vector machines – Kernel: linear; SVM type: C-SVC; Library: LibSVM [9]
Decision tree – Criterion: gain ratio; Depth: 20; Confidence: .25; Minimal gain: .1
Logistic regression – Kernel: dot; ElasticNet: .8; Regularization: .001; Iterations:10
One-vs-rest – Base classifier: Logistic regression
Naive Bayes – Additive smoothing: 1

tomatic detection and classification of newly created issues, this Spark model instance is executed and the classification label is persisted in the AKM tool.

The end-to-end workflow of the automatic design decision detection and classification is shown in Figure 1. Since the output of the first phase (decision detection) is the input to the second phase (decision classification), high accuracy of the results from the first phase is critical. The decision detection component loads the issues, uses the ML model generated for decision detection, and classifies each issue as either a decision or not a decision class. Next, the classification component takes the identified design decisions and classifies them into different categories using the decision classification ML model.

6 Evaluation

In this section, we present the results of applying different classifiers under different configurations for both decision detection and classification using the labeled dataset. In our scenario, the precision (fraction of automatically retrieved documents that are relevant) is as important as the recall (the fraction of relevant documents that were successfully retrieved). For instance, in case of decision detection, it is necessary that all issues that reflect design decisions are retrieved (high recall) and those issues which are not design decisions should not be automatically labeled as design decisions (high precision). Hence, we measure the accuracy as the F-score [28], which is the harmonic mean of precision and recall.

We evaluated multi-class classifiers namely SVM, Naive Bayes, Decision tree, Logistic regression, and One-vs-rest. Since the logistic regression functionality provided by the Spark APIs cannot handle polynomial labels, it was not used for decision classification but only for decision detection (binary). Split strategies from 90% to 50% and n-grams from one to five were analyzed. First, by varying the n-grams from one to five, we expect that the accuracy will proportionally increase. That is, the use of patterns of words, which preserves the context of those words, should positively influence the accuracy of classification. Second, by decreasing the split percentage from 90% to 50%, the accuracy should decrease substantially since lesser number of documents would be used for training the classifiers. In total, 25 individual runs (5 split strategies and 5 n-grams) were executed for each classifier and the corresponding precision, recall, and F-score were calculated. Finally, the average accuracy (average F-score) based on the arithmetic mean of the 25 individual runs for each of the classifiers was analyzed.

Even though the variation of the configuration parameters, namely n-grams and split strategy need not be considered for validating our hypothesis (cf. Section 2), we believe that the impact of these parameters on the F-score is interesting for researchers and will help practitioners to reproduce the results.

6.1 Results - Automatic Design Decision Detection

The SVM classifier (average accuracy: 91.29%) outperformed Logistic regression (83.43%), One-vs-rest (79.45%), Decision tree (79.18%), and Naive Bayes

Table 3. Decision detection: the confusion matrix for SVM

	True decision	True not a decision	Class precision
Decision	**212**	18	92.17%
Not a decision	22	**219**	90.87%
Class recall	90.60%	92.41%	

(76.04%) classifiers. Since, the tf-idf representation of issues has a high dimensional feature space, sparse vectors, and few irrelevant features due to the data curation process, the SVM outperformed the rest of the classifiers. The maximum accuracy of 94.91% for the SVM classifier was achieved for a larger training set (90% split) with 3, 4, and 5 grams representation and the minimum accuracy of 87.4% with a smaller training set (50% split) and 1-gram settings. The confusion matrix for one specific execution run with 70% split and 3-gram configuration is shown in Table 3. This matrix depicts true and false positives as well as true and false negatives. The true positives (correct classifications) are highlighted on the diagonal of the confusion matrix. The precision for classifying an issue as a design decision is 92.17% and the recall is 90.60%. In addition, the precision for labeling an issue as Not A Design Decision is 90.87% and its recall is 92.41%.

Also, as shown in Figure 4 (a), by reducing the size of the training dataset (from 90% to 50%) the F-score decreases as expected but does not diverge more than 4% points from the average F-score of 91.29%. This indicates that the labeled dataset with 1,571 issues is sufficiently large enough to achieve a consistent F-score. Furthermore, it can be observed that the variation of n in n-gram generation does not drastically affect the F-score. As expected, the F-score is comparatively lower when we do not consider the combination of words (n=1) but the F-score slightly improves in the case of 2-grams and 3-grams. However, there does not seem to be any noticeable variations when n is greater than three.

To sum, by using the linear SVM classifier along with n-gram (n >= 2) representation of words, we can automatically extract design decisions from issues (cf. hypothesis 1 in Section 2). To the best of authors' knowledge, since no similar study exists with benchmarking results, we consider 91.29% accuracy for automatic design decision detection to be encouraging.

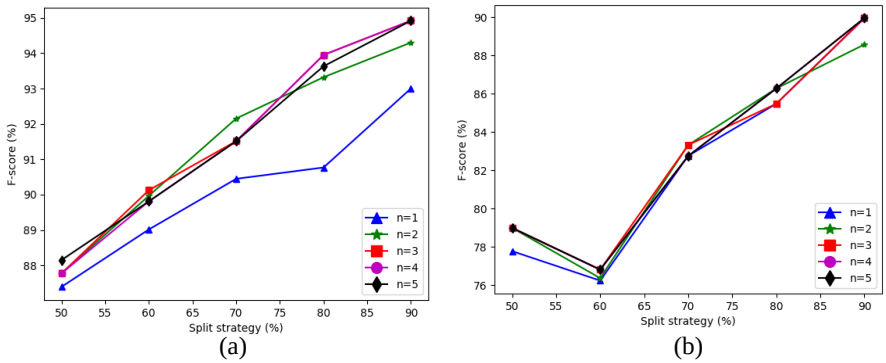

Fig. 4. Influence of n-grams and split strategy on the F-score of SVM: (a) automatic decision detection, (b) automatic decision classification

Table 4. Decision classification: the confusion matrix for SVM

	True ban	True structural	True behavioral	Class precision
Ban	**45**	3	0	93.75%
Structural	4	**41**	13	70.69%
Behavioral	0	6	**39**	86.67%
Class recall	91.84%	82%	75%	

6.2 Results - Automatic Design Decision Classification

Even for the automatic design decision classification, we observed that linear SVM (average accuracy: 82.79%) performed better as compared to classifiers including Naive Bayes (59.09%), Decision tree (60.33%), and One-vs-rest (30%) classifiers. The confusion matrix for linear SVM with 70% training dataset and 30% testing dataset with trigrams is shown in Table 4.

Identifying ban decisions is critical, as they are typically not present in software artifacts (cf. Section 2). As shown in Table 4, the precision (93.75%) and recall (91.80%) for automatically classifying design decisions into ban decisions category are above 90%. On the other hand, the precision for structural and behavioral decisions are 70.69% and 86.67% and their recall values are 82% and 75% respectively. We believe that the lower precision and recall for structural and behavioral decisions is due to the existence of similar features (due to the classification rules presented in Table 1) in their corresponding training dataset.

As shown in Figure 4 (b), reducing the size of the training dataset (from 90% to 50%) decreases the F-score as expected (from 89.9% to 76.2%). This variation is justified since the labeled dataset for decision categories is significantly small (160 design decisions in each category). On the contrary, the variation of n-grams does not have any notable affect on the F-score. This indicates that the individual words within issues (or bag of words in the textual representation of issues) play a significant role in the classification as compared to the usage of specific patterns of words and the context of the words.

To conclude, with the linear SVM classifier we can automatically classify design decisions into structural, behavioral, and ban decision categories with an accuracy of 82.79% (cf. hypothesis 2 in Section 2). However, we perceive that since the dataset for classifying decisions is relatively small, increasing the sample size will improve the generalization capabilities of the classifiers.

7 Threats to Validity

The results presented in the previous section are based on 1,571 labeled issues for design decision detection and 480 labeled design decisions for classification. The labeled dataset for classification is not as comprehensive as the dataset used for decision detection. Even though, we speculate that the generalization capabilities of design decision classification can be further improved by increasing the sample size of the dataset, providing relevant quantitative evidence is beyond the scope of this paper. However, it should be noted that typically in ML-based approaches

for text classification, increasing the sample size of the dataset substantially improves the classification performance [22].

The 1,571 labeled issues are extracted from two large OSS projects, wherein contributors have systematically maintained issues for more than three years. The hypothesis validated using the dataset might not be generalizable for projects where issues are reported scarcely. Hence, understanding what characteristics of the projects could influence the precision and recall of our approach are considered as part of our future work.

In the previous section, we have presented the results of automatic decision detection and classification independently of each other. However, if we consider the workflow described in Figure 1, the accuracy of the decision detection affects the subsequent decision classification phase. In this work, we do not compute the accuracy for the end-to-end workflow. We plan to perform this evaluation after integrating of the workflow within our AKM tool as part of our future work.

Finally, as explained in the data curation process, analysts did not consider issues that could belong to more than one ADD category. Considering such issues would require further investigation into appropriate classification algorithms for multi-label classification and the study of the corresponding results.

8 Conclusion

In this paper, we presented a two-phase ML-based approach to automatically detect design decisions from issues and to subsequently classify them into three ADD categories, namely Structural, Behavioral and Ban decisions. Furthermore, we made the manually labeled dataset used for supervised learning publicly available. This will act as a starting point for researchers to create their own ML models and to compare the accuracy of the automatic design decision detection and classification process. The results presented in Section 6 indicate that we can automatically extract design decisions from issues with an accuracy of 91.29% and classify the extracted decisions into three categories with an accuracy of 82.79% by using the linear SVM classifier. Even though the accuracy can be further improved, we believe that the result is significant enough to demonstrate the feasibility of our approach.

We are currently in the process of integrating our ML pipeline within our AKM tool named AMELIE [4]. This integration will allow us to conduct an extensive evaluation of the ML models in industrial settings. Furthermore, by automatically extracting and structuring design decisions from past projects, we aim to provide recommendations related to semantically similar design decisions in greenfield projects. The process of automatically extracting and classifying design decisions from issues is envisioned to be realized using the end-to-end workflow presented in Figure 1.

To conclude, since design decisions are not explicitly documented but are rather implicitly captured in systems such as issue management systems, automatically detecting, extracting, and systematically structuring them in an AKM tool will help software architects and developers to refer back to already made

design decisions in large-scale software projects as well as in greenfield projects with similar context. Furthermore, classifying them into categories such as Ban decisions will allow stakeholders to reason about those artifacts which no longer exist within the system. Finally, since issues capture both unstructured, as well as structured information, analyzing them, will support the development of decision support systems to address concerns such as "Who took the decision?", "When was the decision taken?", and "Why was the decision made?".

References

1. Ambler, S.: Agile modeling: effective practices for extreme programming and the unified process. John Wiley & Sons (2002)
2. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.G.: Is it a bug or an enhancement?: a text-based approach to classify change requests. In: Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds. p. 23. ACM (2008)
3. Babar, M.A., Gorton, I.: A tool for managing software architecture knowledge. In: Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent, 2007. SHARK/ADI'07: ICSE Workshops 2007. Second Workshop on. pp. 11–11. IEEE (2007)
4. Bhat, M., Shumaiev, K., Biesdorf, A., Hohenstein, U., Hassel, M., Matthes, F.: Meta-model based framework for architectural knowledge management. In: Proccedings of the 10th ECSA Workshops. p. 12. ACM (2016)
5. Bosch, J.: Software architecture: The next step. In: European Workshop on Software Architecture. pp. 194–199. Springer (2004)
6. Buchgeher, G., Weinreich, R.: Automatic tracing of decisions to architecture and implementation. In: Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conf. on. pp. 46–55. IEEE (2011)
7. Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A.: 10 years of software architecture knowledge management: Practice and future. Journal of Systems and Software 116, 191–205 (2016)
8. Capilla, R., Nava, F., Carrillo, C.: Effort estimation in capturing architectural knowledge. In: Proc. 23rd IEEE/ACM International Conf. on Automated Software Engineering. pp. 208–217. IEEE Computer Society (2008)
9. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2(3), 27 (2011)
10. Dagenais, B., Robillard, M.P.: Creating and evolving developer documentation: understanding the decisions of open source contributors. In: Proc. 18th ACM SIGSOFT international symposium on Foundations of software engineering. pp. 127–136. ACM (2010)
11. Fernández, R., Frampton, M., Ehlen, P., Purver, M., Peters, S.: Modelling and detecting decisions in multi-party dialogue. In: Proc. 9th SIGdial Workshop on Discourse and Dialogue. pp. 156–163. Association for Computational Linguistics (2008)
12. Goth, G.: Agile tool market growing with the philosophy. IEEE software 26(2), 88–91 (2009)
13. Hsueh, P.Y., Moore, J.D.: Automatic decision detection in meeting speech. In: International Workshop on Machine Learning for Multimodal Interaction. pp. 168–179. Springer (2007)

14. Jansen, A.: Architectural design decisions. Ph.D. thesis (August 2008)
15. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conf. on. pp. 109–120. IEEE (2005)
16. Kazman, R., Goldenson, D., Monarch, I., Nichols, W., Valetto, G.: Evaluating the effects of architectural documentation: A case study of a large scale open source project. IEEE Transactions on Software Engineering 42(3), 220–260 (2016)
17. Kruchten, P.: An ontology of architectural design decisions in software intensive systems. In: 2nd Groningen workshop on software variability. pp. 54–61. Citeseer (2004)
18. Kruchten, P., Capilla, R., Dueñas, J.C.: The decision view's role in software architecture practice. IEEE software 26(2), 36–42 (2009)
19. Lee, J.: Design rationale systems: understanding the issues. IEEE expert 12(3), 78–85 (1997)
20. Lee, L., Kruchten, P.: Capturing software architectural design decisions. In: Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conf. on. pp. 686–689. IEEE (2007)
21. Lytra, I., Tran, H., Zdun, U.: Supporting consistency between architectural design decisions and component models through reusable architectural knowledge transformations. In: ECSA. pp. 224–239. Springer (2013)
22. Manning, C.D., Schütze, H., et al.: Foundations of statistical natural language processing, vol. 999. MIT Press (1999)
23. Manteuffel, C., Tofan, D., Koziolek, H., Goldschmidt, T., Avgeriou, P.: Industrial implementation of a documentation framework for architectural decisions. In: Software Architecture (WICSA), 2014 IEEE/IFIP Conf. on. pp. 225–234. IEEE (2014)
24. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: Machine learning in apache spark. Journal of Machine Learning Research 17(34), 1–7 (2016)
25. Miesbauer, C., Weinreich, R.: Classification of design decisions–an expert survey in practice. In: ECSA. pp. 130–145. Springer (2013)
26. Porter, M.F.: An algorithm for suffix stripping. Program 14(3), 130–137 (1980)
27. Refaeilzadeh, P., Tang, L., Liu, H.: Cross-validation. In: Encyclopedia of database systems, pp. 532–538. Springer (2009)
28. Rijsbergen, C.J.V.: Information Retrieval. Butterworth-Heinemann, Newton, MA, USA, 2nd edn. (1979)
29. Stettina, C.J., Heijstek, W.: Necessary and neglected?: an empirical study of internal documentation in agile software development teams. In: Proc. 29th ACM International Conf. on Design of communication. pp. 159–166. ACM (2011)
30. Sutherland, J., Viktorov, A., Blount, J., Puntikov, N.: Distributed scrum: Agile project management with outsourced development teams. In: System Sciences, 2007. HICSS 2007. 40th Annual Hawaii Int. Conf. on. pp. 274a–274a. IEEE (2007)
31. Tang, A., Babar, M.A., Gorton, I., Han, J.: A survey of architecture design rationale. Journal of systems and software 79(12), 1792–1804 (2006)
32. Tyree, J., Akerman, A.: Architecture decisions: Demystifying architecture. IEEE software 22(2), 19–27 (2005)
33. van der Ven, J.S., Bosch, J.: Architecture decisions: who, how and when. Agile Software Architecture pp. 113–136 (2013)
34. van der Ven, J.S., Bosch, J.: Making the right decision: Supporting architects with design decision data. In: ECSA. pp. 176–183. Springer (2013)