DEPARTMENT OF INFORMATICS

TECHINICAL UNIVERSITY OF MUNICH

Master Thesis in Informatics

Implementation of an exploratory workbench for identifying similar design decisions

Prateek Bagrecha

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master Thesis in Informatics

Implementation of an exploratory workbench for identifying similar design decisions

Implementierung einer explorativen workbench zur Identifizierung ähnlicher Designentscheidungen

Author:	Prateek Bagrecha
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Manoj Mahabaleshwar M.Sc.
Date:	February 15, 2018

I confirm that this master's thesis is my own work and I have documented all sources and materials used.

Munich, 15.02.2018

Place, Date   Signature

Acknowledgments

First and foremost, I would like to thank my advisor Manoj Mahabaleshwar for his great support and fruitful insights for this thesis. Manoj has been very patience to me and his kindness will always be remembered. Furthermore, I would like to thank Prof. Dr. Florian Matthes for his ideas and for making it possible to write this thesis at the chair of Software Engineering for Business Information Systems held by him.

"Its because I cannot do everything on my own that I have my family and friends"

I have come this far because of the help I received from family and friends. Blessings of my parents and my granny have always stayed with me to remind me every part of the way that I can do this. My brothers and sister-in-law love has supported me throughout.

I would like to thank my friends Akash, Krishna, Pushkar and Kavya for accepting me and pushing me to work hard because it was worth. I thank my friends Koushik, Ajay and Karthik for always being there for me whenever I needed them.

Last but not the least, I would like to thank again my fiancée, Sonal, who despite all my faults has accepted me.

Abstract

Software architecture is a set of high level descriptions or structures that talk about a software system. These structures are required to reason about system design and decisions. Architects and developers regularly make decisions to address stakeholders' concerns during the realization of software systems. Hence, explicitly capturing architectural *design decisions* as a part of *architectural knowledge management* is becoming increasingly important in firms providing professional Information Technology (IT) services.

Large organizations or software projects usually have large number of designs decisions. Often these decisions are similar in nature. However, since the decisions made are seldom shared within an organization, knowledge vaporization occurs. Due to this, reuse of past knowledge to make new sustainable decisions. Furthermore, development of techniques that feasible for deriving similarities between design decisions is a challenging task. Application of clustering algorithms have helped paved way in solving many such challenges, for example, identifying duplicates. Hence, considering this problem to fall under the same domain, the thesis will provide a platform for architects and developers to explore the challenge.

The work here will present a prototypical implementation of a web-based workbench for exploring the use of clustering algorithms and similarity measures to derive similar design decisions in the scope of a single project or multiple. The thesis will provide the necessary information and discuss the need for finding similarities in detail. Moreover, the work here will be using open sources projects that are industrially significant to test the workbench, evaluate the results and provide its limitations.

Table of Contents

List of Figures

List of Tables

Introduction and Theory

Over the past few years, the software research community has emphasized on need for capturing and sharing elements of *software architectural knowledge(AK)* and *design decisions*. One of more recent challenges that researchers are looking to tackle is to prevent vaporization of knowledge related to design decisions. Lately, several meta-models have emerged to efficiently capture and manage design decisions. Many *architectural knowledge management (AKM)* tools use these meta-models to support decision-makers in documenting design decisions. AKMs help in making implicit knowledge residing within a software architecture explicit. As software architecture evolves, the focus is on reusing past design decisions to make new sustainable decisions.

1.1 Motivation

In large software-intensive projects, there are multiple personnel that sustain impairments in terms of time and effort due to analysis of problems that are similar in nature. Software architects and developers often come across scenarios where they must make similar decisions as in the past to address similar design concerns. Lacking awareness of past decisions, architects and developer incur losses in terms of time and money invested when a new design concern arises. The solutions that could be applied across projects are not shared.

In comparing new design concerns with the explicit knowledge from old design decisions, decision-makers can reduce analysis time by inferring details from past decisions. There is a reduction in the time for resolving a new design concern because the turnaround time is also reduced when constraints and design rules are already known.

For instance, Apache Spark is a large opensource software project with more than 20,000 issues that have been captured in an issue management system since early 2014. Much of these issues can be classified as design decisions as they affect the architecture of Apache Spark software. Consider two such design decisions as shown in Table 1. First design decision, DD1, is a decision reflecting a change in the functionality of the system and was created in June 2015. Unaware of the existence of DD1, a similar request DD2 in Table 1 was made in February 2017 in spite of developers having already discussed and closed DD1. DD2 was discussed and resolved in March 2017.

Ref ID	**DD1**	**DD2**
Issue#	SPARK-8321	SPARK-19625
Description	Authorization Support (on all operations not only DDL) in Spark Sql	Authorization Support (on all operations not only DDL) in Spark Sql version 2.1.0
Concepts	Apache, SQL, authentication	Apache, SQL, authentication
Keywords	Spark, operations, Support, Authorization	Spark, operations, Support, Authorization
Components	SQL	Spark Core, SQL
Issue Type	Improvement	Improvement
Created Date	12/Jun/15 03:34	16/Feb/17 09:36
Resolved Date	16/Jun/16 08:22	24/Mar/17 01:21

Table 1 Example design decisions from Apache Spark project

By looking at various attributes of these two decisions, it is evident that they are similar in many ways. They both influence the same set of components: Spark Core and SQL. They are both related to same three concepts: Apache, SQL and Authentication. They have a similar description of an improvement. However, the issues raised are approximately a year and half apart by people with no apparent connection and from different locations. DD2 was resolved as a duplicate of DD1 after a month of discussion. It would have been helpful for the second reporter to have knowledge existence of DD1.

It is clear to see, documenting and sharing architectural design decisions will help architects and developers be aware of past decisions (cf. 2, 3, 9). Documentation helps them in specifying design rules. It also helps them to derive relationship between design decisions and represent them using standard notations such as UML diagrams or directed graphs. These representations form vital means of communication. By using common visual representations, one can then derive complexity involved for addressing similar design concern.

The architects and developers, during the decision-making process, want more information. The information that will help them consider all necessary elements to develop a fault-free maintainable software. This work is motivated by need for making every member of an organization aware of past decisions and avoiding losses mentioned before.

Recent success in using clustering algorithms in the fields of duplicate detection have lead to them be considered in other areas of pattern recognition. The application of the clustering algorithm and similarity measure to detect similar design decisions is a relatively new approach in the domain of software architecture decision-making. Due to lack of previous work in this domain, there is a need for a tool that supports automatic identification of similar design decisions. The primary contribution of this work is to provide a prototypical implementation of a workbench that will help architects and developers explore clustering algorithms and similarity measure for identifying similar design decisions.

1.2 Background

1.2.1 Architectural Design Decisions

This section introduces ISO/IEC/IEEE standard model that defines elements of design decisions.

In the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05) 2005, Bosch and et. al. introduced to the software architecture community a different perspective on software architecture. They defined software architecture as "***a set of architectural design decisions***". [6].

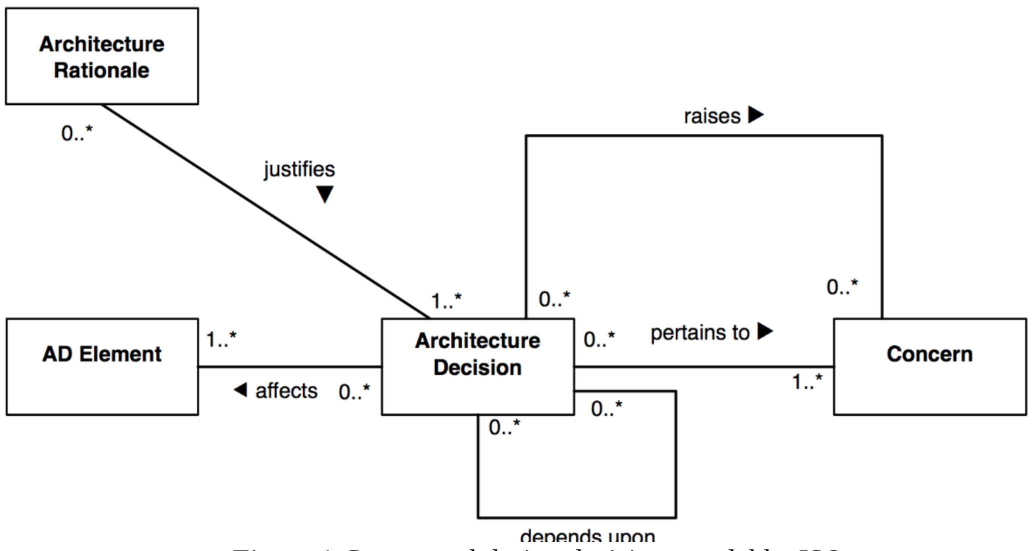
Grady Booch had the following to say about architectural design decisions in his keynote speech at Saturn 2016 Conference "Architecting the Unknown" - ***"In software engineering and software architecture design, architectural decisions are design decisions that address architecturally significant requirements; they are perceived as hard to make and/or costly to change".*** [9]

Informally, an *Architecture Design Decision* is any description that talks about changes to the system, why those changes were made, what behaviours are not allowed and what behaviours are mandatory.

Architectural design decisions are influenced and impacted by the *NFRs: Non-Functional Requirements* of a software systems. They either concern one or more parts of the system or the whole system itself. Each architectural design decision has a description of design concern which is architecturally significant for a software system. Several possible solutions exist for a design concern and Select one of the alternatives in decision-making process.

Currently there are number of models that have been standardized and that define a software architecture as a set of design decisions. Figure 1 shows the conceptual model of Architectural Design Decision and Rationale as standardized in ISO/IEC/IEEE 42010:2011. The work here will use this model to provide an overview of design decisions and concepts associated with it. However, this is used only to understand design decisions and it contributes to this work partially. Later in this work, more models will be discussed to understand on deeper levels.

Figur



1 Standard.

Figure 1 Conceptual design decisions model by ISO

The conceptual model defines three important concepts: *Architectural Rationale, Concern and AD Element*. It also defines all the logical associations between each of them.

AD Element - Any item of a software architecture is considered Architecture Description(AD) Elements. It is recursive definition in the sense that any item that is part of AD element is also consider an AD Element. The changes to a software architecture are nothing but operation that lead to adding, removing or updating the AD Elements.

Concern - Any interest in the system is the "concern" of that system. Examples include purpose of the system, its behaviour, requirements, functionality etc. a design decision pertains to a concern and affects one or more AD Elements. New Concerns may raise by making a design decision.

Architecture Rationale - The justification, or explanation pertaining to design decisions are within the scope of Architectural Rationale. It records why a decision was made and provides a motive for discarding alternative.

1.2.2 Introduction to Document Clustering Algorithms

This section briefly outlines clustering as a technique for finding similarity, similarity measures that are integral for this work. The parts of this section refer to the supplied information to be processed by the algorithms as "*documents*" and the characteristics of the documents as "*features*". However, within the scope of this thesis, the term "document" refers to a design decision and it is used only for purpose of providing generic definitions.

Clustering

Cluster analysis or clustering is the task of grouping a set of documents called as *clusters*. Documents in a cluster are closer to each other than to those in other clusters. This is a very common technique used for recognizing patterns within a set of documents. Various clustering algorithm exists. This works main interest lies in k-Means and Bisecting k-Means clustering, which are discussed in the next sub sections.

Hierarchical Clustering

This is a clustering technique that aims to partition your documents into a *hierarchy* of clusters. There are two strategies in hierarchical clustering: a) Start with each document in its own cluster and merge pairs of them when moving up the hierarchy. b) Start with all documents in one cluster and perform splits recursively when moving down the hierarchy.

k-Means Clustering

This is a clustering technique that aims to partition your *n* document instances into *k* clusters in which each document belongs exclusively to one cluster with nearest mean distance to it. the k-Means algorithm finds groups within documents with number of groups defined by *k*. The algorithm works iteratively to assign each document to one of *k* document groups based on the attributes that are provided. Data points are clustered based on feature similarity. k-means accepts a distance function that it uses to calculate the distance between a chosen cluster centre and a document. The distance function is applied iteratively to all documents in the dataset until all documents belong to a document group or cluster.

Bisecting k-Means

Bisecting k-Means is much like a combination of a hierarchical clustering and k-Means clustering. It first applies k-Means algorithm to create two clusters. This step is called the

“Bisecting” step. It then iteratively repeats the bisecting step and produces more splits and performs all the steps again until desired number of clusters *k* is reached.

Document Clustering

Application of clustering algorithm to textual documents is known as *Document Clustering* (or *text clustering*). This type of clustering technique involves use of *bag of words* that describe the contents of the documents known as *descriptors*. Extraction of the descriptors involve use of many pre-processing operations like tokenization, removal of stop words, stemming etc. Most common use of document clustering is for grouping similar documents such as tweets, news feeds or web context, into meaningful categories list.

k-Means (and its variants) and hierarchical clustering are especially popular for their uses in document clustering. These algorithms can further be classified as hard or soft clustering algorithms. Hard clustering computes a hard assignment meaning each document is a member of exactly one cluster. In soft assignment, each document’s assignment is a distribution over all cluster. In soft assignment, a document has fractional membership in several clusters [19].

In practice, application of document clustering usually involves following steps to be executed sequentially

1. Tokenization

Tokenization is the process of breaking down a document into smaller units called *tokens* such as words and phrases. Documents are input to a *tokenizer*, a program that performs tokenization. Tokenizers identify tokens using methods like regular expressions, flagging, separating sequences using delimiters like comma and semi-colons. This process is called *tokenizing*. Popular methods of tokenizing are bag-of-words, n-gram model and word2vec.

2. Stemming and Lemmatization

To avoid repeated calculation of similar information, we reduce all tokens to their base forms and then group them. This means reducing words to their root forms. For example, grasping to grasp. These processes are called as Stemming (reducing inflected words to their base form) and Lemmatisation (grouping inflected words together).

3. Removing stop words and punctuation

Stop words are tokens are trivial words that do not reveal additional characteristics of a text. It is a good idea to eliminate such words and punctuation marks as they are not very useful for analysis.

4. Feature extraction or generation

After Step 3, it is safe to assume that the remaining tokens distinctly reveal characteristics of the documents. These tokens now must be processed in some way to generate *features* that clustering algorithm can work on. In context of this work, *features* are derived values that promises to be non-redundant and informative.

5. Clustering

After the features are extracted from the tokens, they can now be fed to clustering algorithms for forming meaningful clusters.

6. Evaluation and Visualization

Finally, the clustering models can be assessed by various metrics such f-measure to evaluate its performance and accuracy. It is sometimes helpful to visualize the results by plotting the clusters into low dimensional space.

This ends our introduction to basics of clustering methods and procedure.

1.2.3 Introduction to Word2Vec

Clustering algorithm like k-Means typically require the text input to be represented as a fixed length vector. This kind of representation is very central to natural language processing. The representation of words as sparse vectors derived using training models in neural networks are called *word embeddings*. Word2vec tool is a collection of connected models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space that typically several hundred dimensions in size. Each unique word in the corpus is assigned to a corresponding vector in the space.

Word vectors are positioned in the vector space such that words that share common context in corpus are next to one another. [12]. Word2vec creates vectors that are distributed numerical representations of word features. It does so without human intervention.

Given enough data, usage and contexts, Word2vec can make highly accurate guesses about a word’s meaning based on past appearances. Those guesses can be used to establish a word’s association with other words e.g. “man” is to “boy” what “woman” is to “girl” or “man” is to “king what “woman is to “queen”. It can also be used to cluster documents and classify them by topic. Those clusters can form the foundation of search, sentiment analysis and recommendations in diverse fields of scientific research, legal discovery, e-commerce and customer relationship management.

1.2.2 Introduction to Semantic Similarity and Similarity Measures

This section will introduce you to what semantic similarity is and detail some distance functions or similarity measures that are used in determining it.

Semantic Similarity

Semantic similarity is a metric defined over a set of documents or terms, where the idea of distance between them is based on the likeness of their meaning or semantic content as opposed to similarity which can be estimated regarding their syntactical representation. These are mathematical tools used to estimate the strength of the semantic relationship between units of language, concepts or instances through numerical descriptions obtained.

Similarity is subjective and is highly dependent on the domain and application. For example, two fruits are similar because of colour or size or taste. Care should be taken when calculating distance across dimensions or features that are unrelated. The relative values of each element must be normalized else one feature could end up dominating the distance calculation. Similarities are measured in the range 0 to 1 [0,1].

Similarity Measures

A *Similarity Measure* is the measure of how much alike two data objects are. Similarity measure in context of data mining is a distance between points of dimensions representing features of the objects. If this distance is small, it will be the high degree of similarity where as a large distance will be the low degree of similarity.

A similarity measure is also known as *Similarity Function* which is a real-valued function that quantifies the similarity between two objects. Although no single definition of a similarity measure exists, usually such measures are in some sense the inverse of distance metrics: they take on large values for similar objects and either zero or a negative value for very dissimilar objects.

There two important aspects of the similarity measure that are of used in this thesis viz.

1. **Similarity between two documents or document Vs query terms**: A similarity measure can be used to calculate similarity between two documents, two queries, or one document and one query.
2. **Document Ranking:** similarity measure score can be used to rank how which documents are more similar than others.

All clustering algorithms use similarity or so called "distance functions" to determine cluster members. Few of the most popular similarity measures are discussed in the following subsections.

Euclidian Distance

It is a standard metric for geometrical problems. It is the ordinary distance between two points and can be easily measured with a ruler in two- or three-dimensional space. Euclidean distance is widely used in clustering problems, including clustering text. It is also the default distance measure used with the K-means algorithm. Measuring distance between text documents: given two documents, d_a and d_b represented by their term vectors t_a and t_b respectively. The Euclidean distance of the two documents is defined as:

$$D_E\left(\vec{t_a}, \vec{t_b}\right) = \left(\sum_{t=1}^{m} \left|W_{t,a} - W_{t,b}\right|^2\right)^{1/2}$$

Where, the term set is T = {t_1, t_2,……., t_n}In this calculation $W_{t,a}$ = tf-idf(d_a, t)

Euclidean distance is the most commonly used distance function. In most cases when people talk about distance, they will refer to Euclidean distance.

Manhattan Distance

Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. In a simple way of saying it is the total sum of the difference between the x-coordinates and y-coordinates.

Suppose we have two points A and B if we want to find the Manhattan distance between them. We just have to sum up the absolute x-axis and y-axis. variation means we have to find how these two points A and B are varying in X-axis and Y- axis. In a more mathematical way of saying Manhattan distance between two points measured along axes at right angles.

In a plane with p1 at (x1, y1) and p2 at (x2, y2), Manhattan distance = |x1 – x2| + |y1 – y2|

This Manhattan distance metric is also known as Manhattan length, rectilinear distance, L1 distance or L1 norm, city block distance, taxi-cab metric, or city block distance.

Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. Cosine similarity metric finds the normalized dot product of the two documents. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two documents. The cosine of 0° is 1, and it is less than 1 for any other angle.

It is thus a judgement of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1]. One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors.

Jaccard Coefficient

Jaccard Coefficient is applied to when you want to find similarity between two objects that are sets. It is used to measure similarity between sets, and it can be calculated by dividing the size of the intersection by the size of the union of the sets.

A *set* is (unordered) collection of objects {a, b, c}. The notation of elements separated by commas inside curly brackets {} is used for sets. They are unordered so {a, b} = {b, a}. *Cardinality* of a set A, denoted by |A|, is the count of number of elements in A. *Intersection* between two sets A and B is denoted A ∩ B and reveals all items which are in both sets A, B. *Union* between two sets A and B is denoted A ∪ B and reveals all items which are in either set.

The Jaccard Coefficient measures the similarity between finite sample of sets and is defined as the cardinality of the intersection of sets divided by the cardinality of the union of the sample sets. Suppose you want to find Jaccard similarity between two sets A and B it is the ration of cardinality of A ∩ B and A ∪ B. Jaccard Similarity: J (A, B) = A ∩ B/ A ∪ B

1.3 Research Questions

In this section of the thesis the main research questions that are to be answered are deliberated. Each subsection of the research question will outline the problem domain and provide an approach which will be elaborated in span of the thesis.

What are the functional and non-functional requirements of a workbench that supports in identifying similar design decisions?

Existing systems are analysed in the later part of the thesis to determine what makes a system that supports identifying similarities between design decisions. Important questions such as what the drawbacks of the existing systems are, how does the workbench overcome these drawbacks and additional purposes that are fulfilled by the workbench are discussed.

How to identify similar design decisions using context-aware similarity measures and clustering analysis?

Direct comparison of the design decisions is not possible due to its textual nature and presence of large number of strings. Within this thesis, use of operations that transform design decisions into representations that can be used by similarity measures and clustering algorithms are discussed.

How can a workbench support end-user in identifying the contextual parameters that are necessary for identifying similar design decisions?

Design decisions need to be broken down to smaller parts that convey its architectural description. Only parts that include context information are kept and others discarded. Identifying what parts to be used to identify similarities is of vital importance. Context include a precise knowledge contained within the design decision. The thesis provides a brief reasoning for using only certain parts of the design decision to identify the information contained in it. This question deals with choosing the right similarity measure to compare design decisions. Choice of similarity measure should also be based on the efficiency of the similarity measure in calculating the similarity in context of software architecture.

1.4 Organization of Thesis

Given sufficient motivation behind the topic, necessary background information and the research questions that are going to be answered, the thesis will move to the section to discuss the previous works in the field that have influence the topic. The subsequent chapters are organized as follows: Chapter 3 will provide detailed explanation of experimental setup. Chapter 4 provides an overview of elicited requirements of prototypical implementation of the workbench. In Chapter 5 demonstrates how the prototype has been developed, challenges faced and what are its limitations. Chapter 6 provides evidence of usefulness of the workbench for not only identifying similar design decisions, but also expresses the surprise of additional benefits of obtaining relationship between them. Finally, the thesis will conclude by summarizing the work, mentioning few lessons learnt and providing a future outlook on revision of workbench and clustering techniques used here.

Related Works

This section of the thesis brings to the attention two things: First, a view of evolution of the software architecture models with the lack of information on relationship between design decisions. Second, it summarizes the previous works in the field and how these works have inspired new theories.

2.1 History of Architectural Design Decisions

Until 2004, the definition of software architecture did not mention *Architecture Rationale.* It was first mentioned in the definition provided by Perry and Wolf [18]. Jansen and Bosch in 2005 [6] introduced the software architecture as a composition of a set of design decisions. After which, several papers came to be published on architectural design decision. the community saw a raise in them momentum of research on architectural design decisions.

In 2009, Tang et al. [7] classified architectural knowledge into four broad categories, namely context, design, general, and reasoning knowledge. The context knowledge captures the project-specific information such as management information and architectural significant requirements. The design knowledge comprises of the architectural designs of the software systems.

In 2010, a conceptual model that described relationship between architecture, design decisions and architecture rationale was standardized by ISO/IEC/IEEE board. (cf. chapter 1.2).

In 2016, Bhat et al. [3] introduced a new AKM framework that considered design knowledge to be part of context knowledge and combinedly refer to them as dynamic knowledge. They created a dynamic knowledge model (cf. section 2.2) that overcame the drawbacks of static models in terms of reusability and ability to be configured according to project context.

In 2017, Bhat et. al developed a tool to automatically extract design decisions from issues management system. This tool become part of their AKM framework [8].

So far, the community has not seen any research related to finding or exploring the relationship between two design decisions. However, there are many researches that formalize and define a meta-model to describe design decisions, which has helped in defining a process to identify similarity between design decisions. One such model was already introduced in section 1.2.1, chapter 1.

Few works have pondered as far as finding relations between the issues that are tracked using software like Atlassian Jira, GitHub etc. Some the papers like "Who Should Fix This Bug?" by John Anvik et al go as far as to categorize the text content of the issues and not venture into defining relations between the issues itself [4 & 20]. Also, many papers treat this aspect of the software architecture as classification problem [8] in the sense that they only deal with classifying a design decision into a certain category based on information contained within the design decisions. They do not explore possibility of relationship between the design decisions within a single category. Few papers like “Automated Duplicate Detection for Bug Tracking Systems” by Jalbert et al provide us hints for applying clustering for detection of the duplicate documents.

The subsections following this one will discuss the contribution of few of the aforementioned related works that this work was inspired from. The contributions in this chapter can be categorized into types: Ones that contribute to extracting and identifying duplicate documents and the ones that have applies machine learning to similar fields other than design decisions. Section 2.2.

2.2 Dynamic Knowledge Model

Most organizations such as Apache and Mozilla support documentation of design decisions using off the shelf issue tracking systems (ISMs) like JIRA or Bugzilla. These organizations follow an agile approach in development of a software system. This means the software is delivered with varying architecture as quickly as possible.

AKM framework developed by Bhat et al envelopes the important aspects of the software engineering with respect to issue management systems. This framework thins the boundary between issues and design decision, as the case is nowadays since the introduction to agile methodologies. The dynamic model defined in the framework includes elements from aspects of software project from domain, business, expert work force and activities involved. This is a vital approach as architectural design decisions are influenced by various factors like requirements, time available, human resources, project context and other past decisions. The model was created after analysing the data models of specific issue management system mentioned before. Hence, the model is reflective of modern issue management systems.

As mentioned in previous paragraph, the dynamic knowledge has imbibed within it the data models of issue management systems and Hence, it is not only composed of the architecture management concepts but also concepts of implementation, project management and requirement management. The notion of architectural concern is being perceived differently in this model compared to ISO definition seen earlier in section 1.2, (cf. Figure 2). Architectural concerns are being regarded as *requirements* of a software system. It is safe to assume that these are synonymous of each other. This notion is important as concern is more conceptual term and requirement is a concrete description of a concern.

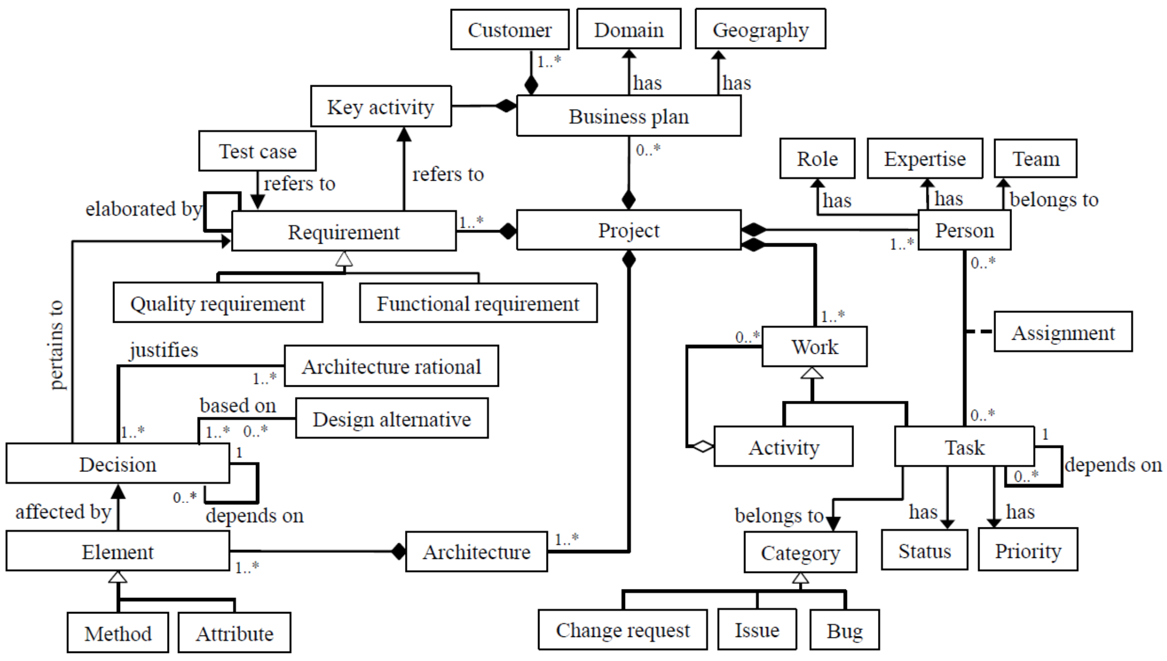


Figure 2 Dynamic Meta Model to capture AK as defined by Bhat et al. [3]

Notice in the meta-model that through the project context every issue can be traced back to one or more requirements. The core concept of the dynamic knowledge model is *Project* that includes essential attributes like such as name, description, etc. A project has multiple requirements and architectural design decisions pertain to these requirements. From the dynamic knowledge model, the following statements can be made:

- A *Decision* affects 1 or more *Architectural Elements*.
- A *Decision* is justified by 1 or more *Architectural Rationale*.
- A *Decision* is based on 0 or more *Design Alternatives*.
- A *Decision* depends on 0 or more other *Decisions*.
- A *Decision* pertains to one or more *Quality Requirement*s.

The dynamic knowledge model is an instance of meta-model hence it can be adopted to project needs at runtime.

The dynamic knowledge provides an abstract representation for both the issues and design decisions and includes necessary elements, wherein the potential context for comparing design decisions could lie.

2.3 Automatic Extraction of Design Decision from Issue Management Systems

Previous section introduces the dynamic knowledge model as developed by Bhat et al. This section describes how that model was used to build a system that automatically extracted design decisions from the issue management systems. The system used machine learning based approach to automatically detect design decisions from issues and to subsequently classify them into three design decision categories, namely Structural, Behavioural and Ban decisions. The labelled dataset resulting from this system has been made publicly available. Figure 3 describes the end-to-end workflow of the pipeline for extracting design decisions from issues.

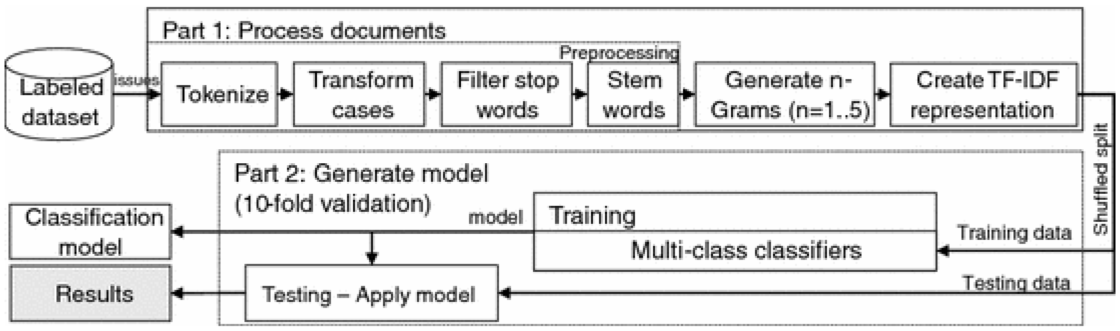


Figure 3: ML pipeline for detecting design decisions and classification by Bhat et al [3].

The pipeline was implemented using Apache Spark's MLlib (machine learning library), which provided interfaces to create and execute the 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 automatic detection and classification of newly created issues, this Spark model instance is executed, and the classification label is persisted in the AKM tool.

Success of such a pipeline-based approach is evident in [10]and the authors have provided a starting point from where the work of this paper takes off.

2.4 Automatic Duplicate Detection in Issue Tracking System

Deduplication of issues is one of the major challenges of organizations that track large number of issues in day-to-day basis. Often, the deduplication process is manual in nature and requires large investment of time by the developers. Jalbert et al in "*Automatic Duplication Detection for Bug Tracking System*" [20] explored the application of clustering algorithms to automatically detect and classify duplicate bug reports. The work used document clustering using textual similarity to classify whether the bug reported is a duplicate or not. The work provided the evidence required to prove that clustering techniques can be applied for textual analysis. The key take-away from this work is that the semantic information is rich in textual descriptions and summary of the issues.

Requirement Analysis and Experimental Setup

This section describes experimental setup that helped in deriving requirements of the workbench for identifying similar design decisions.

As first steps, 1574 issues were extracted from two popular open-source projects (Apache Spark and Apache Hadoop Common) and manually labelled as either reflecting a design decision (784) or not a design decision (790). The design decisions were described used tables with columns and rows, rows representing a design decision and columns representing an architectural concern they pertain to or architecture elements they affect.

Next, the tools for exploring the design decisions were researched and compared. Rapid Miner (RM) studio and WEKA were evidently the most popular and robust tools for beginners to experiment with machine learning techniques. They have workbench that helps users in exploring applications of machine learning techniques. They both contain necessary set of libraries and support materials required for exploring application of clustering algorithms to identify similar design decisions. However, there are few differences that make WEKA a little unfavourable for the thesis. First, WEKA require the data to be provide in ARFF. Secondly, WEKA's data import library was unable to extract design decisions and threw complex errors that were difficult to debug. Finally, WEKA workbench UI is not user-friendly and has a steep learning curve. Due to these reasons and due to the time constraints, a decision was made to use RM studio for initial requirement analysis.

RM studio provides a canvas for designing workflows for data scientists. You can visually create flows that determine how the data is consumed, transformed and have algorithm applied on them to determine patterns in data. Helps in rapid prototyping to experiment with your datasets. It also provides necessary visualization tools to view your results, which makes it attractive for in depth analysis of clusters and similarity techniques. Building blocks of RM studio are called *Operators* and a workflow of interconnected operators is called *Process*. RM studio comes with many operators grouped by their functions, which makes it an ideal tool for rapid prototyping and building quick experimental setup.

Before the dataset can be analysed, essential parts were needed to be extracted to only include required context for the design decisions. This step was necessary to make sure that all the semantically-rich elements of design decisions are included in the clustering analysis. To determine which elements of design decisions have semantically-rich information, a model was derived from the dynamic knowledge model introduced by Bhat et al [4] (cf. Section 2.2) and is shown in figure 4.

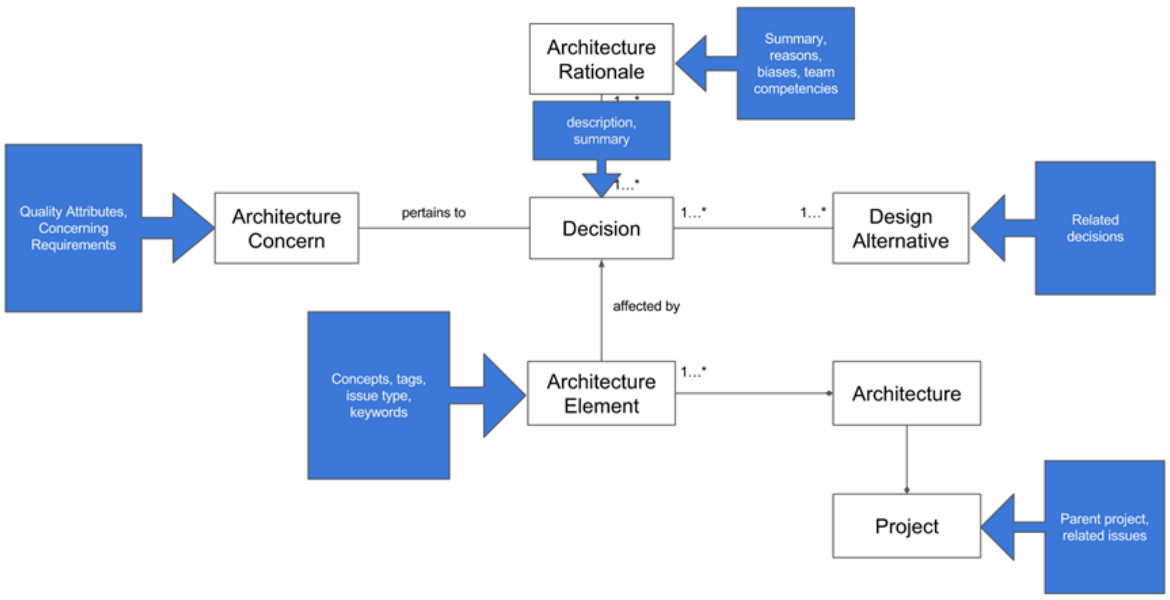


Figure 4: a derived model from dynamic knowledge model

Section 2.2 & figure 2 gave an insight into the relationship between design decisions and the other elements of the dynamic model. The derived model, with its indications of the type of the information that present within it, aids in determining the elements that are rich in semantic information. These elements and their concrete structures when included in the dataset provide adequate material that represents the context of the design decisions. Additionally, they provide the following important inferences that can be considered as the guidelines to detect the similarity between design decision. Any two design decisions are similar if

- They *affect* 1 or more same *architectural elements*.
- They *justified* by 1 or more similar *architectural rationale*.
- They are *based* on same set of *design alternatives*.
- They *depend* on 0 or more other similar *design decisions*.

- They *pertain* to one or more similar q*uality requirement*s or *architectural concern*.

Using the above inferences, the corresponding the attributes of the issues were extracted and the dataset was built. For instance, looking back at design decisions mentioned in Table 1, we can infer that DD1 is similar to DD2 because they both affect component SQL, or DD1 is similar to DD2 because they both share same set of concepts: Apache, SQL, and Authentication.
Three initial experimental setups with RM studio had meant to reveal answers to two challenges. 1) how to apply clustering analysis on textual data with missing attributes? 2) which similarity measure are suitable for determining similarity between textual data?

RM studio provides parameters *measure_types* and *mixed_measure*. These parameters allow users to specify the type of measure to user for the input dataset. For textual analysis, the measure type must be mixed measure as design decisions may contain combination of textual and categorical data. The *mixed_measure* parameter allows us to set the similarity measure to use and this is set to 'Mixed Euclidean Distance', this is the only available option.

First setup was to iteratively apply a similarity measure to pairs of all design decisions. RM studio provides an operator called "Data to Similarity" to execute this approach. The parameters were set to type mixed measure and Mixed Euclidean Distance. The approach had inconclusive results (see figure 5) as every design decision was found to be equidistance from every other. The second approach was also inconclusive because it led to non-inform cluster formations.

Second setup used K-Means with Mixed Euclidian measure. In this process, the leading design decisions were initially assigned to a cluster each and consecutive design decisions were assigned to the first cluster with exception of design decisions that had missing values. The design decisions with either empty summary or empty description were grouped together in their own clusters (see figure 6).

The initial experiments clearly revealed a necessity for application of pre-processing operations on the design decisions like replacing missing value and removing punctuation on the datasets the workbench should convert them into formats that is comparable by distance measures and later for determining the similarity between them.

Final approach was to add pre-processing steps to the workflow right before distance calculation. So, the datasets should be uploaded in particular format, pre-processed using some natural language processing technique, converted to vectors for numerical representation and clustering algorithm was applied to determine clusters of similar design decisions. This workflow was particularly useful in classification of textual data as seen in previous works mentioned in Section 2 and expressly seen in [10]. The workflow is especially useful when here is a need to apply same set of pre-processing steps in both classification of design decisions and to identify similarities between them.   Results of this can be observed in figure 7,8 and 9.

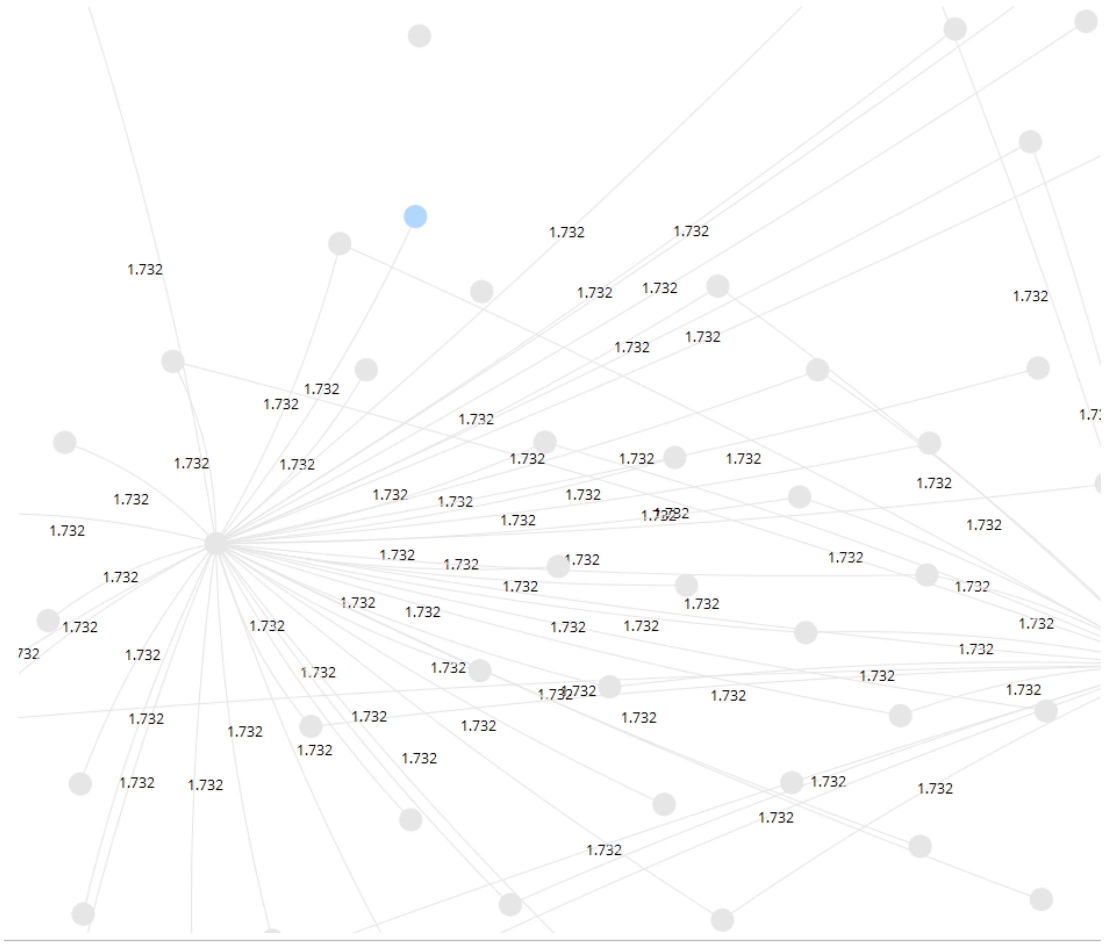
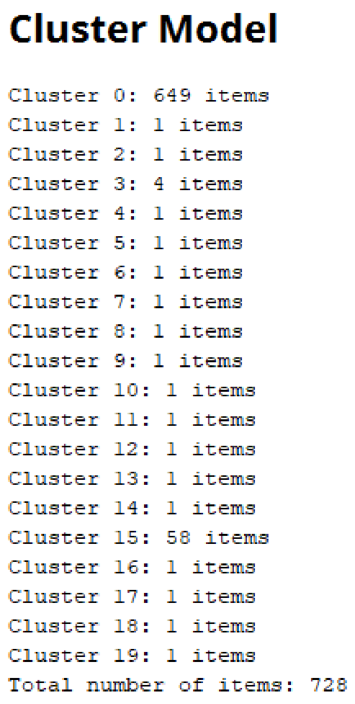


Figure 5: Comparison of Design Decision directly using Mixed Euclidian Similarity Measure

The experimental setups discussed above helped in identifying the essential operations that could be applied to the design decisions, the operations that make them readable for analysis in clustering algorithms. Different capabilities of RM studio and WEKA aided in identifying the initial set of requirements of a prototypical workbench that will allows end-user to identify similar design decisions (cf. Chapter 4).



Cluster Model

```
Cluster 0: 649 items
Cluster 1: 1 items
Cluster 2: 1 items
Cluster 3: 4 items
Cluster 4: 1 items
Cluster 5: 1 items
Cluster 6: 1 items
Cluster 7: 1 items
Cluster 8: 1 items
Cluster 9: 1 items
Cluster 10: 1 items
Cluster 11: 1 items
Cluster 12: 1 items
Cluster 13: 1 items
Cluster 14: 1 items
Cluster 15: 58 items
Cluster 16: 1 items
Cluster 17: 1 items
Cluster 18: 1 items
Cluster 19: 1 items
Total number of items: 728
```

Figure 6: Results of comparing design decision directly using Mixed-Euclidian similarity measure

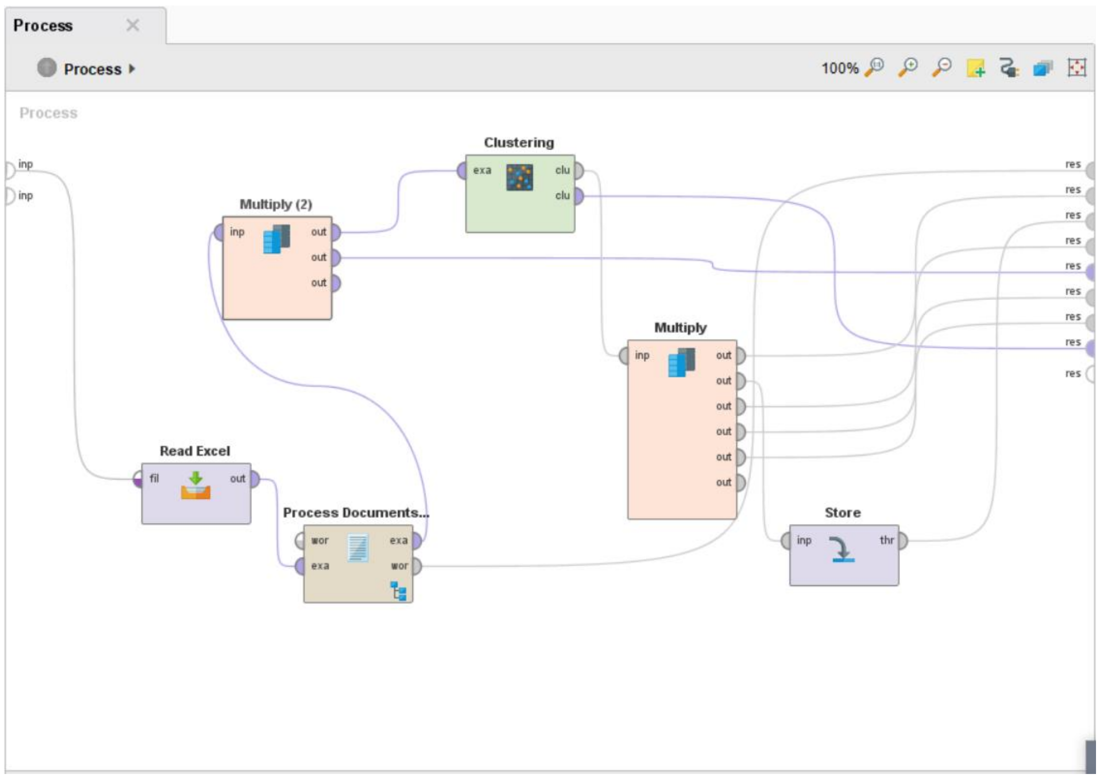


Figure 7: Sample clustering process from RM Studio

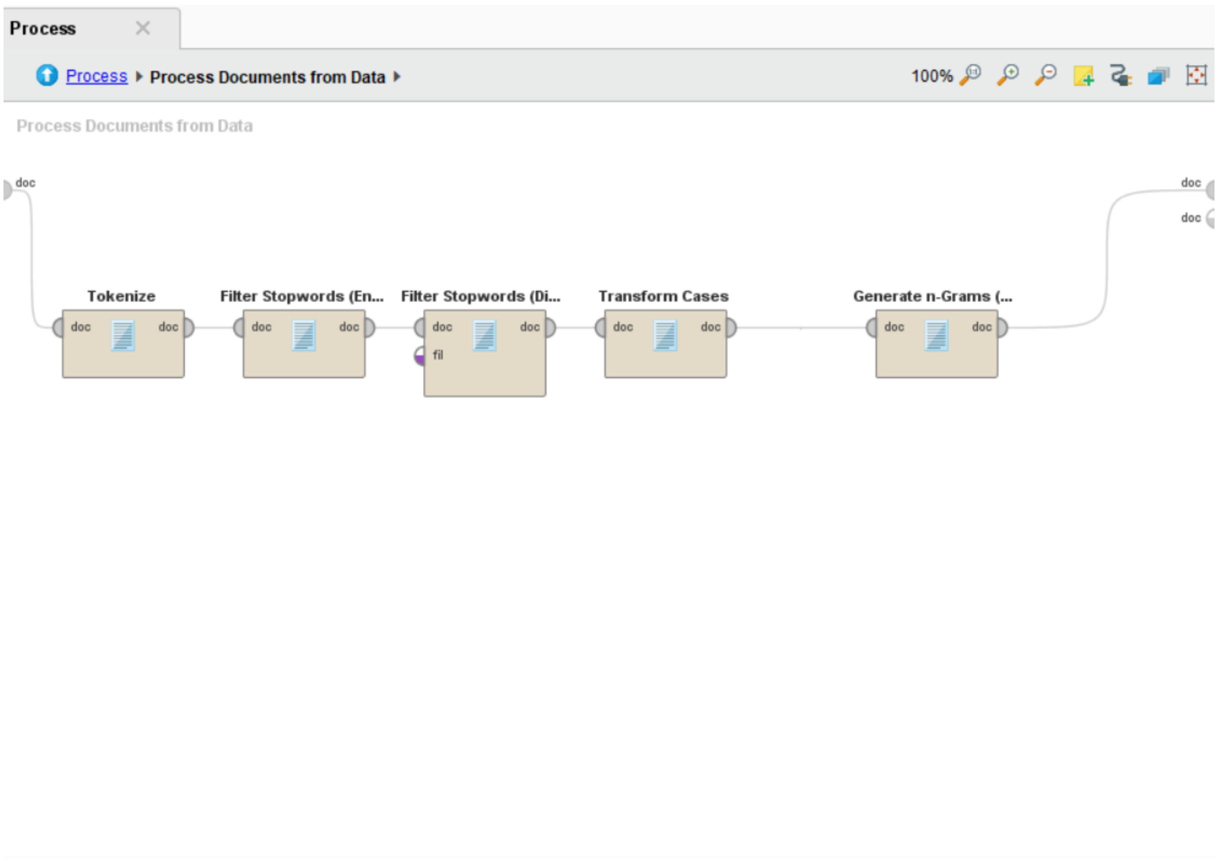
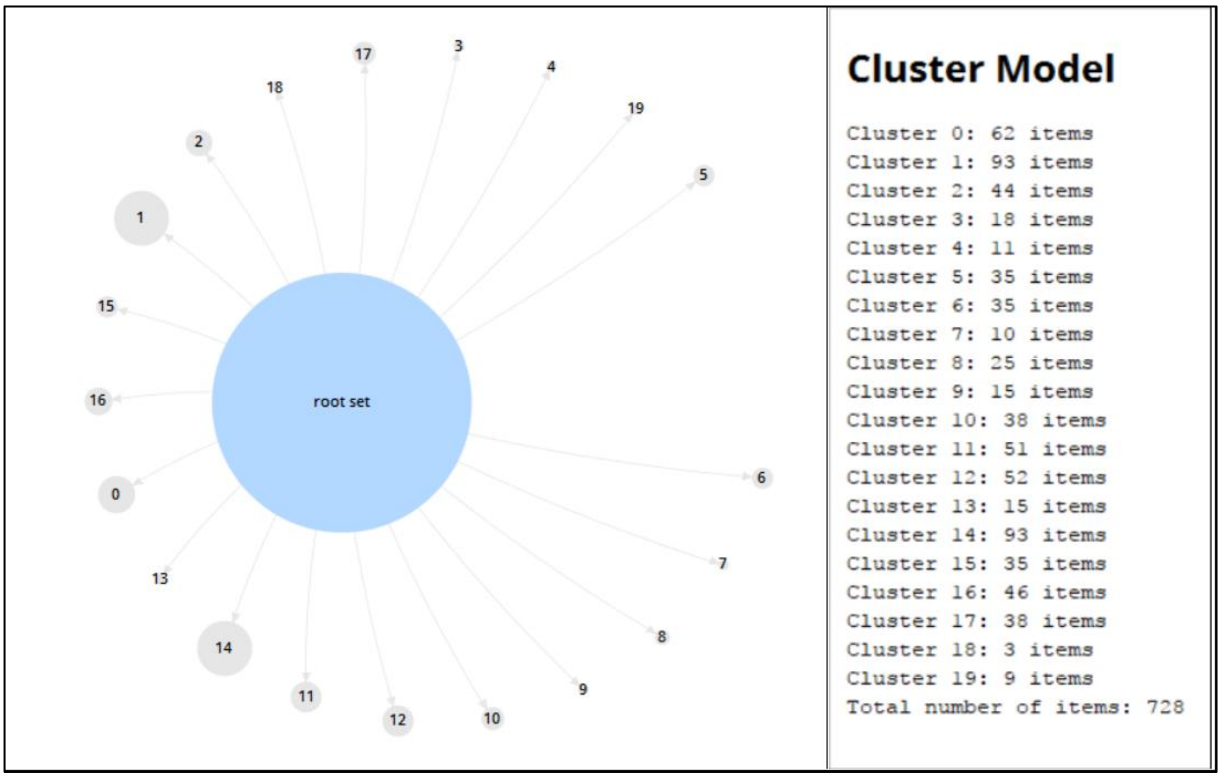


Figure 8 Sample pre-processing steps from RM Studio



Cluster Model

```
Cluster 0: 62 items
Cluster 1: 93 items
Cluster 2: 44 items
Cluster 3: 18 items
Cluster 4: 11 items
Cluster 5: 35 items
Cluster 6: 35 items
Cluster 7: 10 items
Cluster 8: 25 items
Cluster 9: 15 items
Cluster 10: 38 items
Cluster 11: 51 items
Cluster 12: 52 items
Cluster 13: 15 items
Cluster 14: 93 items
Cluster 15: 35 items
Cluster 16: 46 items
Cluster 17: 38 items
Cluster 18: 3 items
Cluster 19: 9 items
Total number of items: 728
```

Figure 9: Sample results from RM Studio

Requirements

This section lists the requirements gathered after the analysis of experiments conducted and explained in Chapter 3. Apart from the results gathered from the experimental setup, the current process and architecture of AKM framework developed by Bhat et al (cf. Section 2.3) and their AKM tool AMELIE (cf. Appendix) were also analysed. Overall function of AMELIE is described in figure 10. The AMELIE system imports issues from JIRA, extracts design decisions from these issues, annotate the architectural elements in the extracted design decisions, label the quality attributes that issues pertains to and provides recommendation for new design decisions about experts.

The prototypical workbench implemented was made to fit well into the existing ecosystem of the AMELIE and hence had to fulfil certain functional and non-functional requirements that are described as follows:

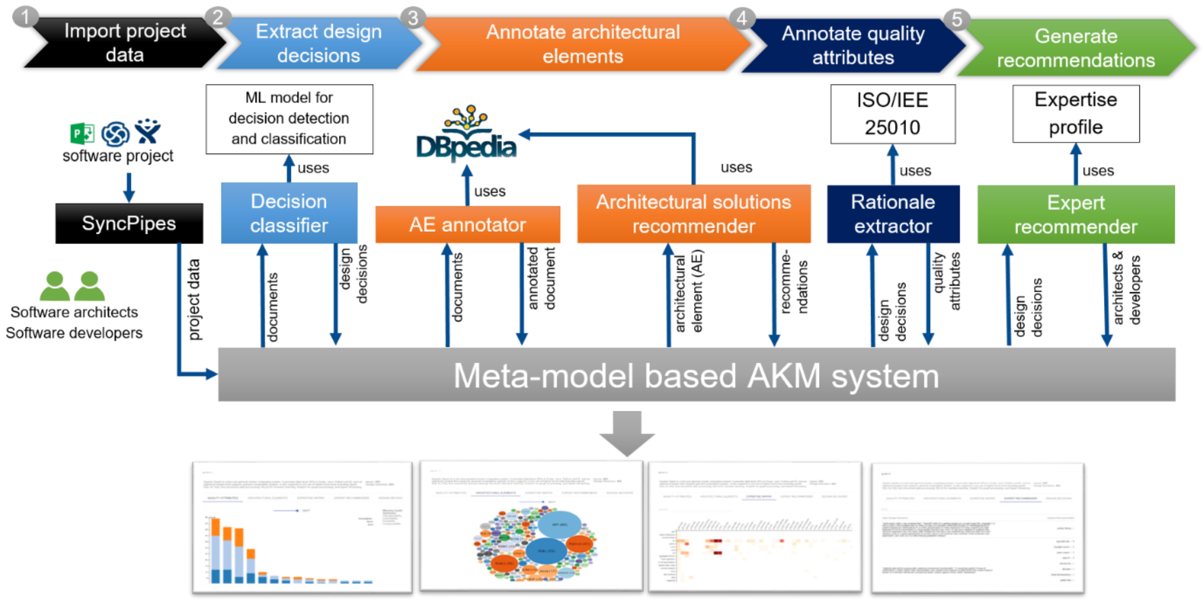


Figure 10: AMELIE System Architecture

Requirement 1: Workbench must provide necessary tools to the user to explore different clustering algorithms and similarity measures

The workbench shall be an exploratory web application in the sense it should be a web site that allows users to create and reuse multiple experiments. The ability to create experiments by input of different parameters by the users for exploring different clustering algorithms and similarity measures.

Requirement 2: Workbench must be configurable

The workbench shall be configurable in the sense that a person holding a copy of the workbench shall be able to spin off his own version of workbench by having an ability to turn on and off the features of the workbench.

Requirement 3: Workbench must provide both a user interface and RESTful APIs

End-users with the help of either a user interface or RESTful APIs should be able to create and configure multiple experiments for clustering design decisions and to input new design decisions to predict similar past decisions.

Requirement 4: Workbench must have the ability to automatically import date from SocioCortex and AMELIE knowledge based

End-user should be able to link and import data from the entities and sub entities of the SocioCortex and AMELIE Knowledge base.

Requirement 5: Workbench must be able to consume different data formats

The workbench shall be able to consume more than one format of data like csv, json etc.

Requirement 6: Workbench shall abstract all operations related to identifying similar design decisions

The operations configured by the end-users should run in background as a single batch. The end-users see only the abstracted execution of the experiment and only see information that is absolute for them.

Requirement 7: Workbench must provide reusable components

The workbench's architecture should foster reuse of components. Following the object-oriented approach, maximum classes and objects of the workbench shall made available for reuse to extend the framework when deemed necessary by the end-user. Within the workbench, it should be easy to integrate and to interchange foreign components. Any external libraries or extensions written for the libraries shall be available in such way as to they can be switched with other component of similar nature without effecting the readiness of the system.

Requirement 8: Workbench shall provide extensions point for integrating multiple machine learning libraries

Many machine learning libraries exists now, and every end-user has his/her choice of weapon. Workbench must encompass this philosophy into it and provide points or interfaces that allow extending its capabilities.

Overview of Workbench Architecture

This section provides overview of workbench's pre-requisites, process flow, and system design. System design is discussed in the last part of this section to provide an easy mapping between it and the fulfilled requirements.

5.1 Pre-requisites

As first steps, for a given project that maintains issues in an issue management system, the end-users extract all the existing issues into an architectural knowledge management (AKM) system AMELIE (c.f. Section 4, Figure 10). Next, the decision detector component of AMELIE identifies those issues that reflect design decisions using a supervised machine learning (ML) algorithm. For each identified design decisions, the end-user can proceed to store it back to SocioCortex or AMELIE for future use.

The workbench is developed for the end-users to explore through the design decisions that have already been made and recorded by them. The workbench assumes that the end-user has a new design concern and wants to see if there exist other past decisions which have addressed this. Before proceeding to use the workbench, the end-user must do following things

1. End-user shall use AMELIE to identify and extract all design decisions
2. The end-user shall know where the data is to be extracted from: does he/she wish to extract from SocioCortex by linking the workspace and entities? or does he/she wish to upload the data in a standard format like csv. (currently the workbench only supports csv data format for uploading)

Provided the end-user has the design decisions, start training the workbench with few clicks (c.f. Section 5.2).

5.2 Overview of Application Concepts

This section discusses the overall conceptualization of workbench and describes elements defined within its architecture. The high-level overview of the process is shown in Figure 5.1.

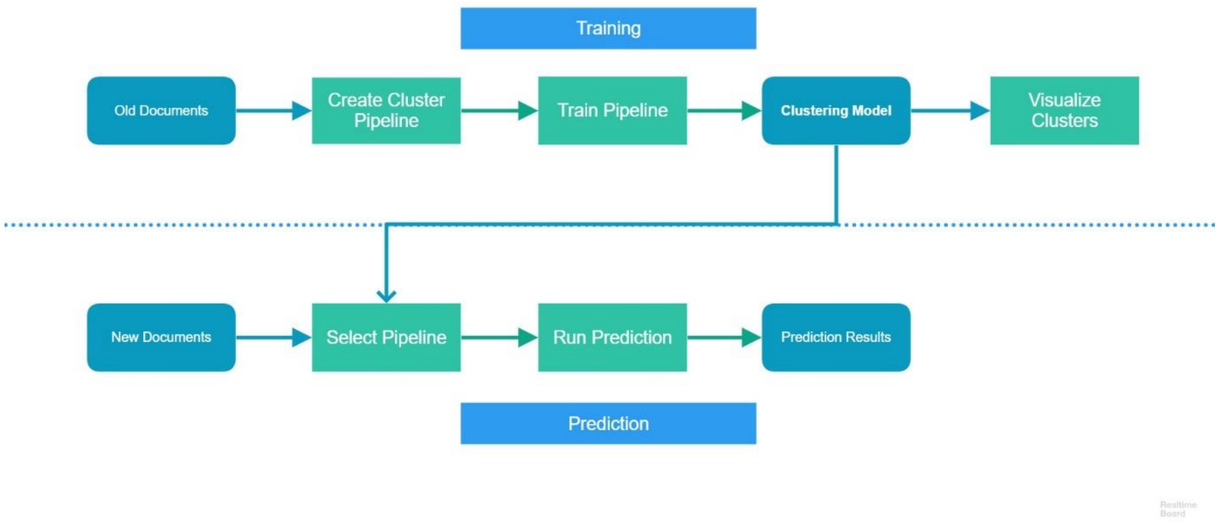


Figure 11: User-centric process overview for training and predict similar design decisions

The workbench is implement based on the requirements that were mentioned in the previous section. End-users will be provided with a web application that will help them explore through their experiments through the concept of workflows. Workflows are automation of group of tasks that are executed either sequentially or parallelly. Each user can create his own workflow for exploring his/her design decisions.

The concept of workflows that are discussed here follow the convention defined by Apache Spark ML library. Apache Spark ML Team defines workflows in a manner that is easy to adopt, it provides a set of abstraction for understanding and applying machine learning algorithms and pre-processing functions.

A typical standard machine learning workflow has following steps:

I. Data Ingestion
II. Features Extraction
III. Model Training
IV. Prediction

Following the Apache Spark convention, the pipeline's key concepts are following

1. Pipeline
2. Pipeline Stage

3. Transformers
4. Estimators
5. Models

Pipeline

As discussed earlier, a *Pipeline* is a single workflow within the context of machine learning. Creating a pipeline means defining stages of the user defined operations that are to be performed on the design decisions. A pipeline consists of one or more *Pipeline Stages*.

Pipeline Stage

A *Pipeline Stage* represents a single stage in a pipeline and is an operation that is applied to each of the design decision. A pipeline stage operation could a data transformation function such as removing punctuations or applying a ML algorithm. A pipeline stage can either be a Transformer or an Estimator.

Transformer

A *Transformer* is a pipeline component that transforms a document into another document after applying some user defined function on it. User defined functions are usually applied to the documents to prepare before a machine learning algorithm can work with it. An example of a transformer would be a user defined function that changes all capital letters in the documents to lowercase.

Estimator

An *Estimator* is a machine learning algorithm that learns from the documents, the processing is known as fitting a model. An estimator produces a *Model* for a given set of documents and parameters, which is then used to calculate predictions. An example of an estimator would be K-Means clustering algorithm.

Model

The Model produced by an Estimator is a Transformer by itself that transforms the input documents by adding predictions to them. The process of generating a model on applying a learning algorithm is called as fitting the model. Depending whether the model was fitted during the training phase or the prediction phase a model can either be Pipeline Model or Prediction Model.

Every pipeline is an instance of typical ML workflow. A pipeline has one or more (usually more than two) pipeline stages. A stage implements can a transformer (that can load data or extract features) or an estimator (that performs ML). The design decisions fed by the user to workbench are passed through each pipeline stage in the order defined. The outputs of a pipeline execution are a model that can be used later for predicting similar design decisions and a cluster table of design decisions.

5.3 User-Centric Pipeline Execution Flow

The entire process of training and identifying the similar design decision is divided into two parts: *Training* and *Prediction* (which is detailed in the Section 5.3) and their corresponding types of pipelines are known as *Training Pipeline* and *Prediction Pipeline*. As mentioned in the section 5.2, pipelines are typical ML workflows and the workbench allows users to quickly create, assemble and configure workflows that are instances of pipelines that can be executed.

At any point of time users can either create a new training pipeline that trains a pipeline model or a new prediction pipeline, which is a pipeline created using an already trained model.

Figure 11 shows the process diagram that the user will follow to create and train pipelines. Figure 5.3 shows the homepage of the workbench, which displays the workflow overview of training and prediction pipelines.

Training Pipeline

In context of machine learning and in our application, training means generation of a model that uses the existing design documents to learn patterns emerging within the design decisions. The model transforms the input design decisions into a group of design decisions that are assigned to clusters based on emerged patterns.

The created training pipeline follows the steps in training phase from figure 11 when it is executed. End-user creates a pipeline with the necessary configuration. The workbench currently supports creating a pipeline with following configurations: Name of pipeline, Library to use, clustering algorithm to use, clustering parameters, selection of feature extractor and uploading documents in some format or linking to a SocioCortex workspace. (see figure 13).

Once the user saves and runs the training pipeline following operations take place: The design decisions are extracted from the provided file or linked workspace, transformations are applied using specified library and configurations, a trained model is generated (eventually saves it) and resulting clustered are display as table and graph.

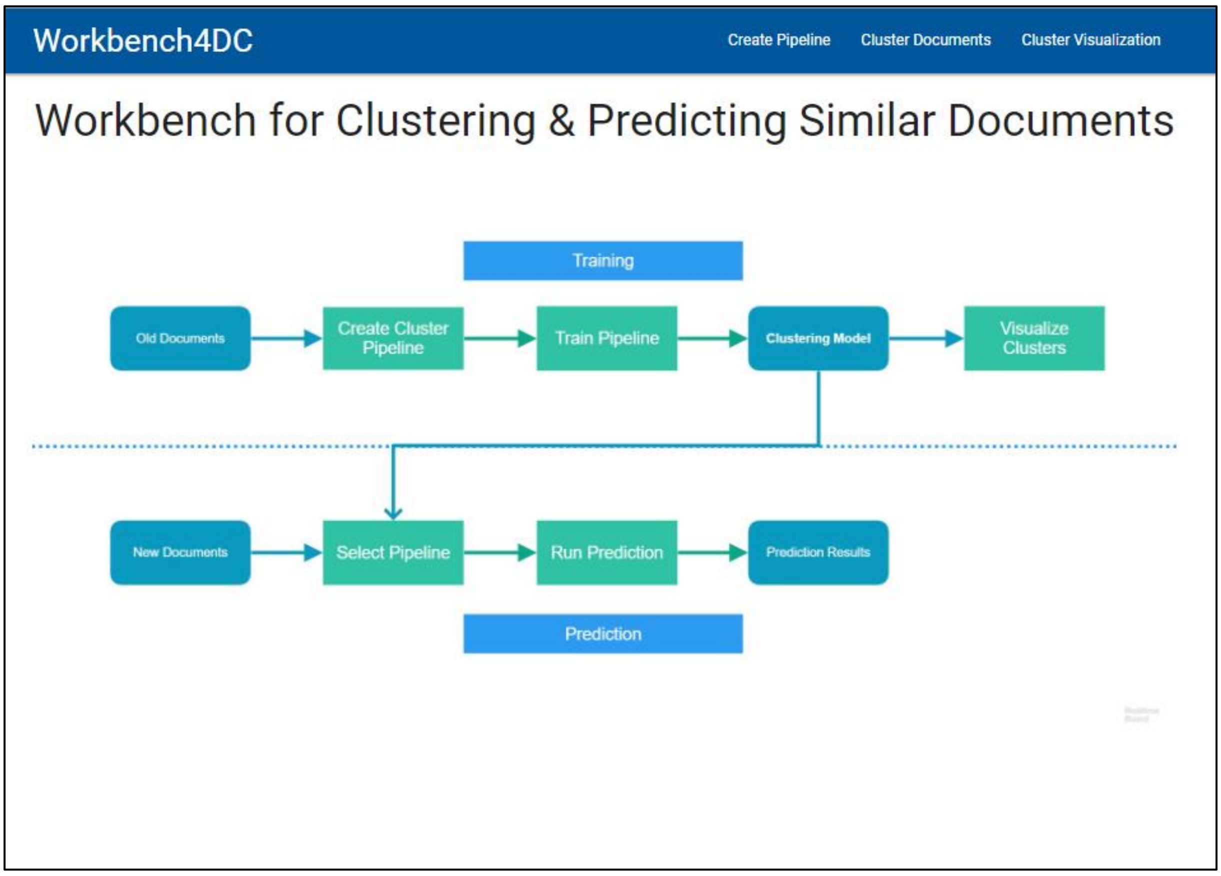
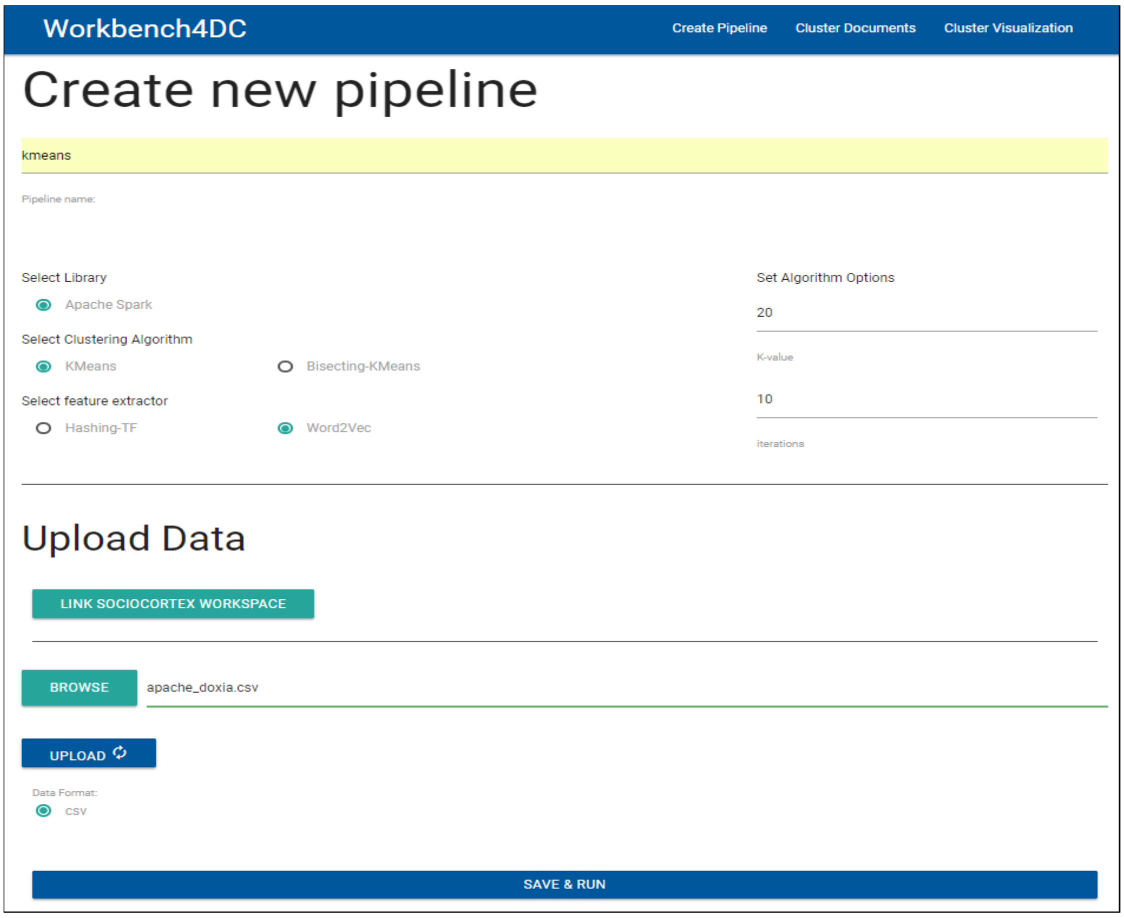


Figure 12 Homepage of Exploratory Workbench for Document Clustering and Prediction

When creating a pipeline, end-user must choose the library he wants to use, transformations to apply to the design documents, select ML algorithm and upload the design decisions. After the design decisions are input and before the transformations are applied, the workbench will first proceed to clean the data provided to it. The cleaning process involves digesting data using multiple functions and this is called as pre-processing stage. The current pre-processing functions applied by the workbench are concatenating all string attribute values, removing punctuations and converting all string values to lowercase. After the pre-processing stage, pipeline stages are executed. Once all the transformers are applied, marking the completion of pre-processing stage, the design documents are ready to be fed to the machine learning algorithm. Estimators are applied, and the Model is fitted (see figure 14). From here the workbench proceeds further to transform example design decisions using the model generated to obtain a pre-calculated set of clusters. The number of cluster is defined by the end-users. These pre-calculated set of clusters provide us a smaller set of design documents to compare.

Prediction Pipeline

Prediction means transforming the given new design decisions using the model previously obtained in the training phase and predicting a cluster label for the provided design decisions. Prediction pipeline *predicts* the cluster that for the new design decisions falls under and identifies decisions with similar context.



Workbench4DC — Create Pipeline | Cluster Documents | Cluster Visualization

Create new pipeline

Pipeline name: kmeans

Select Library: [✓ Apache Spark]

Select Clustering Algorithm: [✓ KMeans] Bisecting-KMeans

Select feature extractor: Hashing-TF [✓ Word2Vec]

Set Algorithm Options

K-value: 20

iterations: 10

Upload Data

LINK SOCIOCORTEX WORKSPACE

BROWSE apache_doxia.csv

UPLOAD

Data Format: [✓ csv]

SAVE & RUN

Figure 13 Page for Creating Training Pipeline

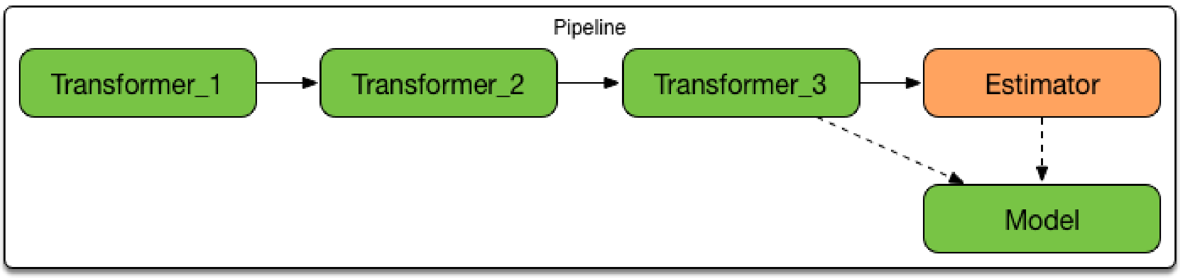
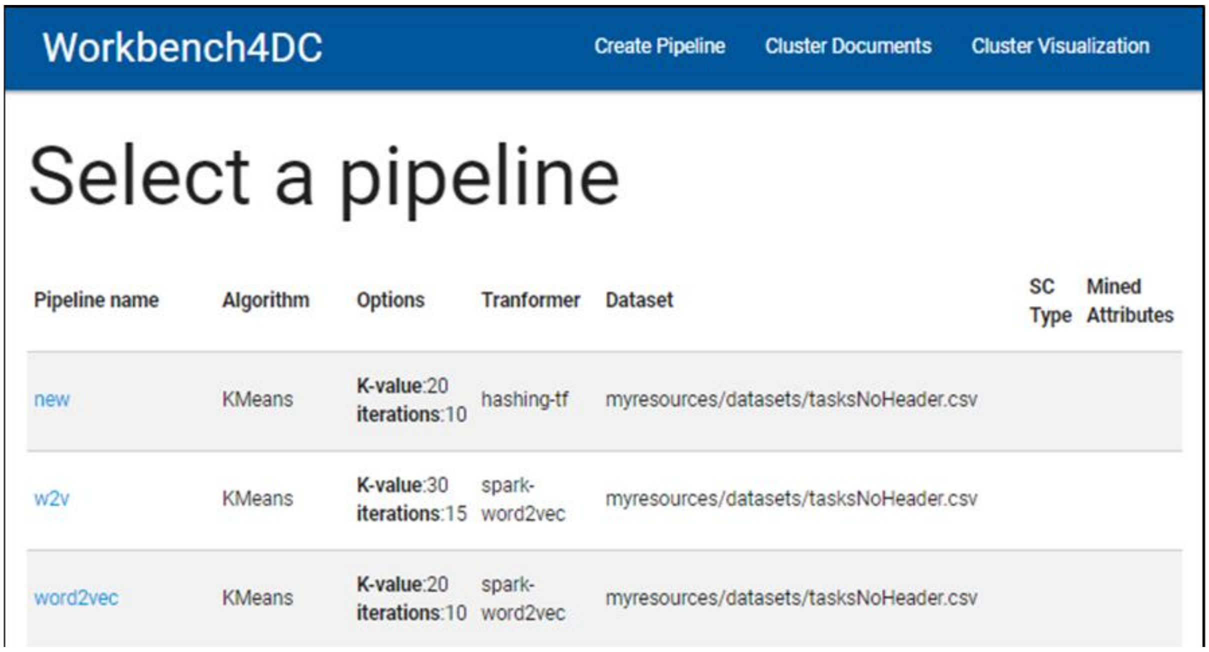


Figure 14 Training Pipeline Stages Example

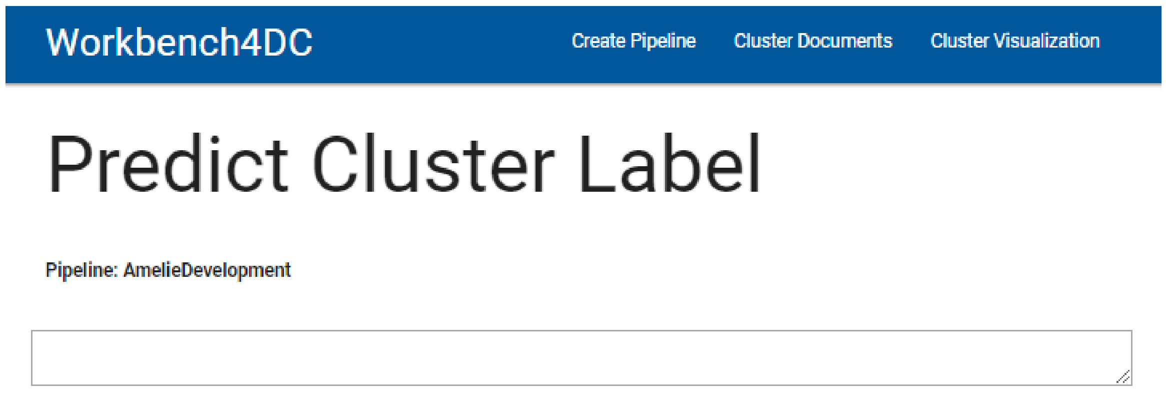
A prediction pipeline follows the steps in prediction phase from figure 11 when it is executed. In the prediction phase, user selects a previously trained pipeline to run with a new design decision. (see figure 15 & figure 16). Once the prediction pipeline runs, it predicts the cluster label for the given design decision. The cluster label is then used to retrieve past design decisions that are part of the same cluster. Next, iterative comparison is made between each of the past design decision with the new design decision using similarity measure cosine similarity and Jaccard coefficient. This stage is the ranking stage. The result of the ranking stage consists of degree of similarities, which can then be used to determine the top design decisions that are similar (see figure 15).



Select a pipeline

Pipeline name	Algorithm	Options	Tranformer	Dataset	SC Type	Mined Attributes
new	KMeans	**K-value**:20 **iterations**:10	hashing-tf	myresources/datasets/tasksNoHeader.csv		
w2v	KMeans	**K-value**:30 **iterations**:15	spark-word2vec	myresources/datasets/tasksNoHeader.csv		
word2vec	KMeans	**K-value**:20 **iterations**:10	spark-word2vec	myresources/datasets/tasksNoHeader.csv		

Figure 15 Application Page showing a list of previously trained pipelines



Predict Cluster Label

Pipeline: AmelieDevelopment

Figure 16 Application page showing the selected pipeline and text area to add new design decision

5.4 System Architecture

This section outlines the design of the workbench, providing details of essential components that make up the architect of the system.

The system is divided into three layers: *Persistence, Middleware and Presentation*. Persistence layer handles the storing and retrieval from storage and has components specific to the type of storage. The middleware layer handles the training design decisions and predicting results. The presentation layer has views that are exposed to the end-users. The architecture is realized using carefully chosen languages, and frameworks based on the requirements. Next sections will discuss in detail the components and the roles of the components present within each layer in detail. The overview of the complete architecture with its components is shown in figure 17.

Persistence Layer

The workbench uses two types of storage: the MongoDB and Physical filesystem. It uses MongoDB component to communicate with a Mongo database. The *MongoDB* component is responsible all storing and retrieving operation from the MongoDB. The workbench stores the configurations of libraries (that it is extended with) and pipelines created by the user in MongoDB. The trained models and results are stored within the physical filesystem.

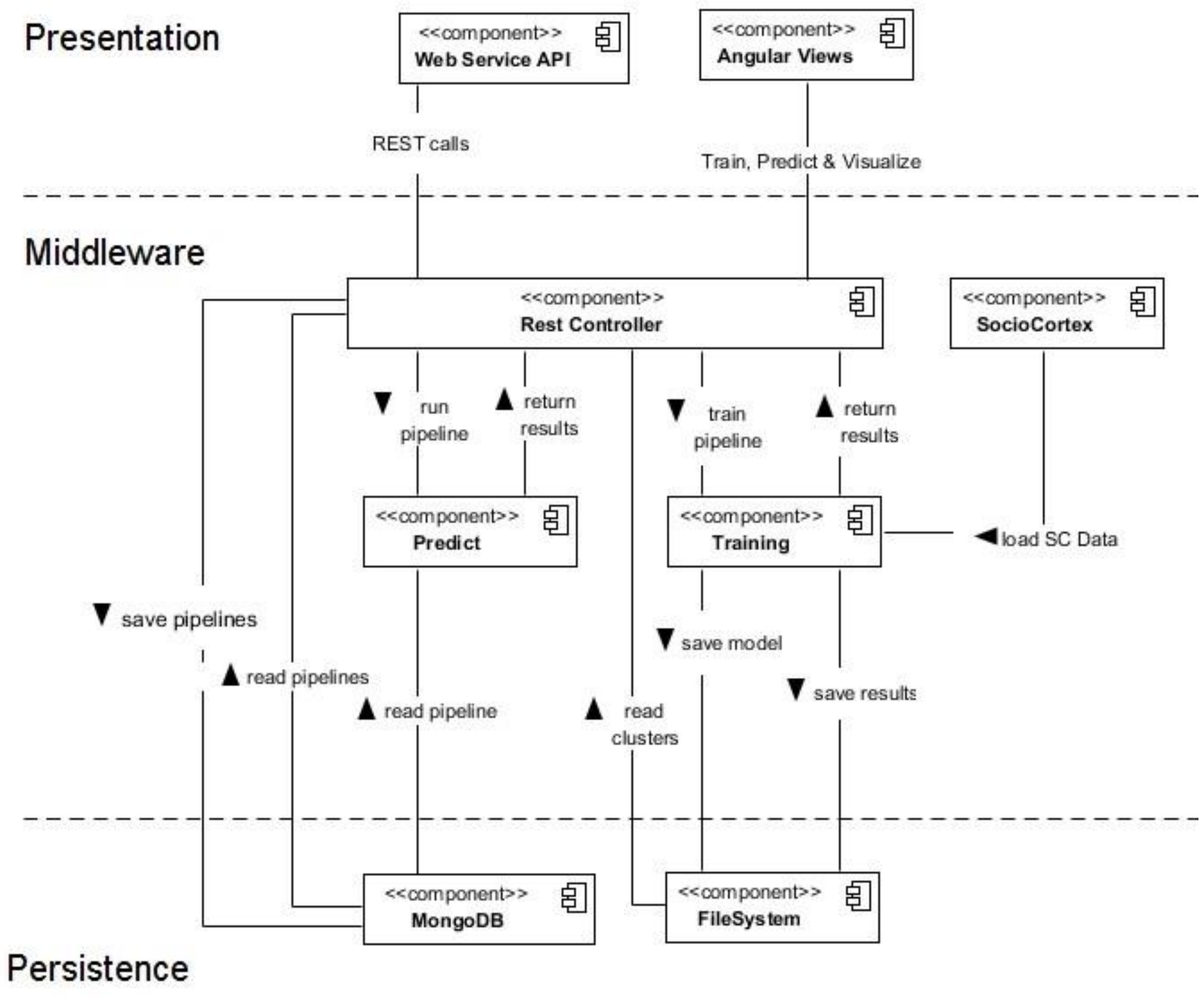


Figure 17 System architecture of the workbench

Middleware

The middleware is the core of the workbench, where all operations are carried out. It is composed of three main components and other supporting components: Predict, Train and REST controller.

Training

Training components encapsulates the implementations of all tasks dealing with training the pipeline. It has supporting components that allow clustering operations. It consists of a main factory class that creates and stores pipeline configurations. The factory class creates objects of the classes that implement interfaces of a training pipeline. a sample UML view as show in figure 18. The IDataLoader interface is also provided to facilitate extending the framework to add other format data handling.

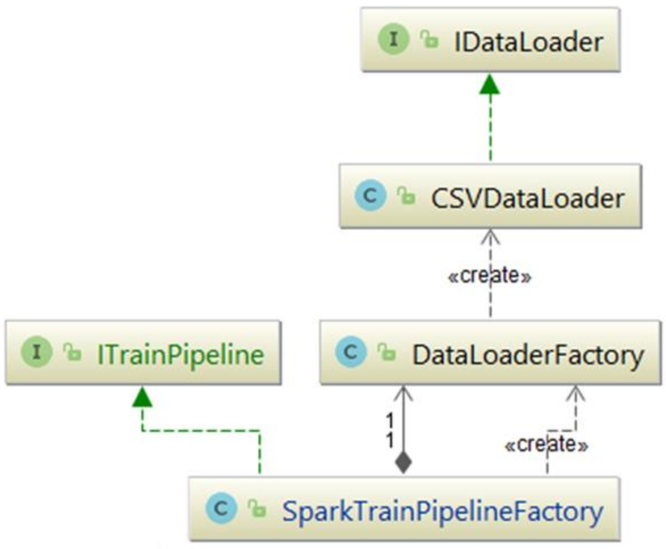


Figure 18 Sample class diagram of training component

Predict

The predict component is responsible for all operation dealing with predicting the cluster label for a new design decision and generating results containing similar design decisions. It consists of a class that uses IDataLoader interface to load models and results into it. Its main component is the *Ranking* component that applying similarity measures to the clusters and returns percentage similarity between provided new design decision and the other cluster members. Sample UML diagram of the Predict component is as shown in figure 5.9. Currently only cosine similarity and Jaccard similarity are implemented.

REST Controller

All incoming requests are channelled to the middleware through this component. The REST controller uses the defined routes to forward requests to the right component. The routes are defined in a config file called *routes.conf*.

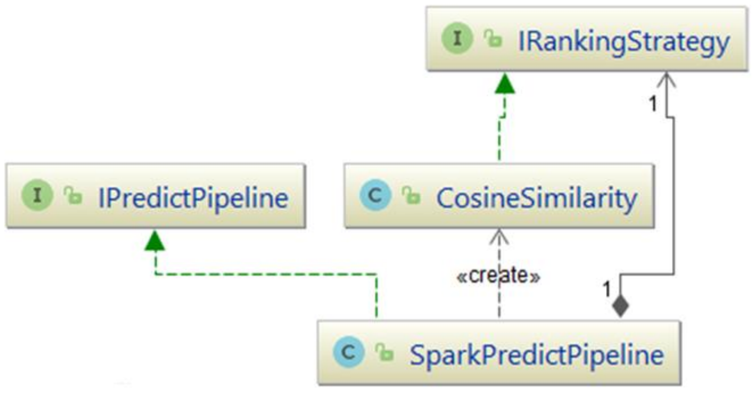


Figure 19 Class diagram of predict component

Presentation Layer

The presentation layer presents the users with a web-based interface that lets users train and predict similar design decisions. The views have been implemented in AngularJS and presents the user with necessary forms for training and predicting design decisions (c.f. Section 5.3, figures 5.2 - 5.7).

Additional components apart ones mentioned in these layers exists. However, they have been omitted to mention here as they are basic components are part of languages and frameworks and thesis assumes its audience have basic language of these. Additional, documents required for efficient handover are either provided in appendix or in code repository.

Review and Assessment of Requirements

This section will discuss the achieved requirements (c.f. Chapter 4) for the workbench.

Requirement 1: Workbench must provide necessary tools to the user to explore different clustering algorithms and similarity measures

Requirement 1 has been achieved since user can explore many different options for clustering using the workbench with help of pipelines. Users can create as many experiments as he/she wants and run them.

Requirement 2: Workbench must be configurable

The workbench uses MongoDB and physical filesystem to store the configuration that it must run with. Several configurations are available to adjust at any point of time to adjust the workbench settings (can be found in the workbench code base). This requirement has been covered.

Requirement 3: Workbench must provide both a user interface and RESTful APIs

Requirement 3 is achieved by exposing both a UI and web services to the end-users. End user can create and execute pipelines using the REST Methods provided. All necessary REST methods are implemented and are available (c.f. Appendix C API Documentation).

Requirement 4: Workbench must have the ability to automatically import date from SocioCortex and AMELIE knowledge based

This requirement specifies that end-user should be able to link and import data from the entities and sub-entities of the SocioCortex and AMELIE Knowledge base. The requirement has ben met and available in the workbench as discussed in Section 5.3 & 5.4.

Requirement 5: Workbench must be able to consume different data formats

Due to the time constraints, the workbench has not been extended to support other data format imports yet. However, adequate interfaces are to extend the framework with more data loader components and classes. This requirement is marked incomplete and will be taken up as part of future works.

Requirement 6: Workbench shall abstract all operations related to identifying similar design decisions

With the help of pipelines, the end-users are hidden from the implementations and execution of the training and prediction. End-user only has to concern himself with the design decisions that he is feeding and nothing else.

Requirement 7: Workbench must provide reusable components

All components that are necessary throughout the application are added to separate directories (corresponding to the library directory or parent directory) and are available as static classes and methods. Requirement 7 has been marked as complete.

Requirement 8: Workbench shall provide extensions point for integrating multiple machine learning libraries

Requirement 8 has been achieved by providing adequate interfaces and factory methods are provided to facilitate extension of the workbench features.

Other than these high-level requirement, more function and non-functional requirements have been recorded. A table of requirements is added to the appendix for reference.

Evaluation and Results

The traditional evaluation strategies for clustering involve splitting the datasets in to training and testing datasets. The clustering model is trained on the training dataset and the evaluated against training dataset. However, the use of clustering algorithm is to identify emerging patterns and areas of concentrations for datasets. However, the workbench only provides users with ability to create clustering models with their set of configurations. The purpose of the workbench is to provide ability to run multiple experiments using end-user's strategy and integrate its results into his/her own AKM framework. For this reason, the thesis here employs an evaluation strategy that simply answers the question: does such a workbench work? And what are its additional benefits.

To evaluate the workbench to see if it gets relevant results, first, many sample datasets with varying number of design decisions were required. The test datasets needed to be created which contained mixtures of duplicated and related design decisions. Such test datasets were creating by scheming through various open source projects and identifying design decisions within them. Next, design decisions that affected other design decisions in different projects were collected. The obtained datasets were mixed together other non-duplicates and irrelevant design decisions. This was to evaluate if the workbench found other design decisions that are could have been of interest. The datasets obtained were as follows: Two open source projects: Apache Solr and Hadoop Commons, one dataset from AMELIE (already identified design decisions from Apache Spark and Hadoop) and one dataset containing design decisions that affect the SQL component of the underlying systems. Once the datasets were obtained, the were fed to the workbench to train and to identifying similar design decisions. For predicting, the description of a selected design decision (combinations of summary, descriptions and other related attributes) was used. If the workbench, return itself and duplicates, the test was marked as success. The following are evaluation steps,

1. Take a design decision
2. Obtain its relative context values (Summary, Descriptions, Keyword etc.)
3. Feed to predict pipeline
4. From results obtained, verify if its duplicates exist in the same cluster
5. Check its similarity index

The workbench was successful in returning the original design decision and duplicated with high cosine similarity (~90% and above) and Jaccard similarity (~40% and around). Percentage similarity for Jaccard Measure was found to greater than 60% for most duplicates. One surprisingly additional benefit of the workbench was that it also clustered related design decisions within the same cluster and ranked them with high similarities. This is taken as sign of the novelty of the workbench. While most clustering algorithms to find areas of concentrations and classification of the textual document, the workbench uses the clustering algorithm with completed different purpose of identifying related and duplicate issues.

There could only limited evaluation of the workbench mainly due to the low number of design decisions that have affects across projects and inadequate maintenance of the related issues. Feedback from two industrials partners brought into highlight necessity for the workbench is not just for identifying design decisions. It can also be used in recommending experts who could handle new design decisions.

Conclusion

To summarize, the work here provided key insights into application of machine learning algorithm for document clustering analysis combined with the power of similarity measures. The conclusion is drawn based on success of the workbench in not only identifying duplicate design decisions but also related ones. The workbench can also be used to reveal critical flaws in process followed in documenting design decisions.

The requirements fulfilled by the workbench were analysed to see if it's a feasible solution. The workbench three additional benefits from a data engineer's perspective: first, if developed further, the workbench promises to be a single atomic tool all data analysis needs. Second, the workbench framework can also be used for application of document clustering other than design decisions. Third, the workbench exposes external APIs to consume its functions, which most tools do not provide.

Success is but temporary for the workbench needs to develop further to make it a powerful with ability to do supernatural things. The difficulty of implementing a workbench with can cross function with different algorithms was not overlooked. No two ML library are the same. One thing, they have different representation of the models and results. Given below are concerns and reasons that provide insight into limitations of the workbench.

Why two documents within the same cluster could have a negative or zero value for similarity measure?

Clusters are calculated by randomly picking a document to be represented as cluster centre and all other documents are included in the cluster based on their distance calculated by some distance measure from that centre. If a border is visualized around cluster, then the two documents that you try to compare may be present near the opposite side of the border. They belong to the same cluster, however, they are too far apart each other. Hence, they turn out to be very dissimilar.

Problems with Jaccard Coefficient

Jaccard Coefficient calculates similarity for documents based on number of distinct words with in the documents. So, as size of document increases, number of distinct words increases. More distinct words between documents means that there are very less number of common terms between them. Jaccard Coefficient becomes very small to imply that two documents are dissimilar, however that might not be case if you look at the documents closely sometimes.

Few more reason why Jaccard Coefficient fails is due to existence of words that are rarely used together such as words that are joined (ex. Class names, object variables) etc. there is set defined way to separate out these kind of words as the way the words are joined may be on purpose or my mistake.

You say workbench is customizable. Yet I see only one library, why?

- Configurable means here one can create a pipeline with any configuration
- Workbench is configurable in sense, we have laid down the foundation for you to extend the workbench with multiple libraries. A clear defined flow is present with good UI.
- Libraries with come with their own set of limitations: Have their own way to defining results, tailoring them to fit the result into a generic one takes a lot of time. Hence, I decided to concentrate on a single library but to build at least a complete Clustering-Prediction workflow.

Why Spark?

- Open Source, no license issues unlike Rapid Miner
- Spark consists of its own SQL library, meaning SQL syntax for easy data operations, retrieving datasets, selecting rows etc.
- Apply iterative operation on all rows without too many for loops

Limitations with Spark

Spark library does not have any implementation of stemming, but good news is that the workbench can be extended to include your own version of stemming.

Bibliography

[1] Managing architectural decision models with dependency relations, integrity constraints, and production rules by Olaf Zimmermann, Jana Koehler,Frank Leymann, Ronny Polley and Nelly Schuster, 2008.

[2] ISO/IEC/IEEE 42010:2011,Systems and software engineering - Architecture description, the latest edition of the original IEEE Std 1471:2000,Recommended Practice for Architectural Description of Software-intensive Systems (www.iso-architecture.org/ieee-1471/cm/).

[3] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, Michael Hassel, and Florian Matthes. 2016. Meta-model based framework for architectural knowledge management. In Proccedings of the 10th European Conference on Software Architecture Workshops (ECSAW '16). ACM, New York, NY, USA, Article 12, 7 pages.

[4] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug?. In Proceedings of the 28th international conference on Software engineering (ICSE '06). ACM, New York, NY, USA, 361-370.

[5] Olaf Zimmermann, Christoph Miksovic, and Jochen M. KüSter. 2012. Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services. J. Syst. Softw. 85, 9 (September 2012), 2014-2033.

[6] Jansen, Anton; Bosch, Jan -Software architecture as a set of architectural design decisions, Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on 109-120, 2005, IEEE.

[7] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar. A comparative study of architecture knowledge manag. tools. J. of Syst. and Soft., pages 352-370, 2010.

[8] Slimani, Thabet. (2013). Description and Evaluation of Semantic Similarity Measures Approaches. International Journal of Computer Applications. Vol 80. 25-33. 10.5120/13897-1851.

[9] Grady Booch, Architecting the unknown, Saturn 2016

[10] Bhat, Manoj & Shumaiev, Klym & Biesdorf, Andreas & Hohenstein, Uwe & Matthes, Florian. (2017). Automatic Extraction of Design Decisions from Issue Management Systems: A Machine Learning Based Approach. 138-154. 10.1007/978-3-319-65831-5_10.

[11] Levy, Omer and Yoav Goldberg. “Dependency-Based Word Embeddings.” ACL (2014).

[12] From Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Word2vec

[13] Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.

[14] Bosch J. (2004) Software Architecture: The Next Step. In: Oquendo F., Warboys B.C., Morrison R. (eds) Software Architecture. EWSA 2004. Lecture Notes in Computer Science, vol 3047. Springer, Berlin, Heidelberg

[15] Kruchten, Philippe. (2004). An Ontology of Architectural Design Decisions in Software-Intensive Systems. 2nd Groningen Workshop on Software Variability.

[16] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," in IEEE Software, vol. 22, no. 2, pp. 19-27, March-April 2005.

[17] Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla, and Muhammad Ali Babar. 2010. A comparative study of architecture knowledge management tools. J. Syst. Softw. 83, 3 (March 2010)

[18] Perry, D. E.; Wolf, A. L. (1992). "Foundations for the study of software architecture". ACM SIGSOFT Software Engineering Notes. 17 (4): 40.

[19] Manning, Chris, and Hinrich Schütze, Foundations of Statistical Natural Language Processing, MIT Press. Cambridge, MA, May 1999

[20] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), Anchorage, AK, 2008, pp. 52-61.

[21] Saimadhu Polamauri, April 2015, “Five most popular similarity measures implementation in python, https://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/

Appendix

A. Use Case Scenarios

Use Case ID	UC1
Use Case Name	Create Pipeline
Actors	Architects, Developers, Data Engineers
Description	User accesses the workbench and views the create pipeline page of the workbench. A form is presented to the user to input the required configuration of the pipeline. User is provided a "save & run" to create and execute the pipeline
Preconditions	1. User has navigated to create pipeline page using the top navbar provided
Postconditions	1. System has stored the pipeline configuration to database. 2. System has stored the trained model. 3. System has stored the cluster results 4. User is presented with cluster graph
Normal Flow	1. User navigates to Workbench URL 2. User click on "Create Pipeline" in the Navbar provided at top 3. User enters the pipeline name 4. User selects library 5. User selects ML algorithm to use 6. User selects transformer to use 7. User clicks on "Browse" button 8. User selects the file he wants to upload 9. User click "Upload" button 10. User selects data format option 11. User clicks on "save & run" button
Alternate Flow	7a. In Step 7, User has the option to link SocioCortex workspace. 1. User clicks on "Link to SC Workspace" button. 2. User provides a filename with extension 3. User selects the workspace from drop down 4. System displays all the available entities from SC 5. User selects an entity of the workspace 6. User selects multiple mining attributes 8a. Step 8 is skipped
Assumptions	1. User has already extracted the design decision using AMELIE

Use Case ID	UC2
Use Case Name	Visualize Pipeline
Actors	Architects, Developers, Data Engineers
Description	User accesses the workbench and views the visualize page of the workbench. A table is presented to the user listing all available trained pipelines. On select a pipeline, user is redirect to a page that load cluster graphs and cluster table.
Preconditions	1. User has navigated to visualize page using the top navbar provided
Postconditions	1. System displays user two sections: Cluster Table and Cluster Graph 2. System allows user the ability to expand and collapse part of the graph
Normal Flow	1. User navigates to Workbench URL 2. User click on "Visualize" in the Navbar provided at top 3. System displays a table of previously trained pipelines 4. User selects a pipeline 5. System redirects user to a page containing cluster table and cluster graph
Assumptions	1. User has already extracted the design decisions using AMELIE and trained a pipeline use those design decisions

Use Case ID	UC3
Use Case Name	Predict Pipeline
Actors	Architects, Developers, Data Engineers
Description	User accesses the workbench and selects one of the previously trained pipeline that he wants to predict results from. User is redirect to page containing text, where he provides the description of the design decision. Once user runs the predict pipeline, user displayed a table of similar design decisions
Preconditions	1. User has navigated to cluster documents pipeline page using the top navbar provided
Postconditions	1. System displays to user a table of similar design decisions
Normal Flow	1. User navigates to Workbench URL 2. User click on "Cluster Documents" in the Navbar provided at top 3. System displays a table of previously trained pipelines 4. User selects a pipeline 5. System redirects user to a page containing a text area 6. User inputs the description of a new design into the text area 7. User clicks on “predict” button
Assumptions	1. User has already extracted the design decisions using AMELIE and trained a pipeline use those design decisions

B. Requirements Specifications

Requirement ID	**Title**	**Description**	**Type**	**Priority**
R1	Interface for creating pipelines	System should provide user a form that allows them to create their own pipelines with their choice of library and algorithms.	Functional	1
R2	API for creating pipelines	System should expose a POST Method to create pipelines	Functional	1
R3	Interactive graphs	System should provide graphs to visualize the pipeline executions results at anytime	Functional	2
R4	Expand/collapse facility for view design decisions	System should provide within the graph an expand or collapse capability to view cluster members and design decisions	Functional	3
R5	List all trained pipelines	System should provide a list of previously trained pipelines for visualization and predicting	Functional	1

R6	Graph with non-redundant words with in design decisions	System should present to the user the list of non-redundant words within a design decision.	Functional	3
R7	Interface to predict similarities	System should present the user with a form to enter a new design concern and predicting similarity.	Functional	1
R8	API to predict similarities	System should expose a POST to the user, to which he/she can give a pipeline name and input design decision. Results should be returned to be in json format	Functional	1
R9	Interface to view predict results	After running prediction, System should mention the cluster the design decisions falls under and display a table with calculated similarities	Functional	2
R10	Order in prediction results	System should display the similarity results in descending order of one of the similarity measure	Functional	2

R10	homepage of the workbench	System should have a homepage that presents user with the overview of the process	Non-Functional	3
R11	API for retrieving clusters	System should expose a GET method to retrieve previously trained pipeline clusters	Functional	2
R12	API for retrieving previously trained pipelines	System should expose a GET method to retrieve previously trained pipelines	Functional	2
R13	API for libraries	System should expose a GET method to retrieve workbench's current configurations	Functional	2
R14	Interface to Link to SC	System should provide necessary forms to link and import data from SC	Functional	1
R15	Interface to upload design decisions	System should provide necessary form elements to upload a dataset with design decisions in certain format	Functional	1
R16	Cluster table view	System should present users with cluster tables with member count within each cluster	Functional	3

R17	Navbar	System should provide a bar on top for users to navigate to pages	Functional	3

C. API Specifications

Title	Method	Path	Parameters
Get all libraries	GET	/clustering/libraries	None
Get all pipelines	GET	/clustering/pipelines/	none
Get a pipeline	GET	/clustering/pipeline/:pipelineName	pipelineName: String
Get all cluster of a pipeline	GET	/pipeline/clusters/:pipelineName	pipelineName: String
Create pipeline	POST	/clustering/pipeline/create	Cluster Pipeline Object that contains following 1. Name of pipeline: String 2. library code: Number 3. algorithm code: String 4. transformers: Object 5. dataset values or minning attrbiutes from SC: String or Array of Strings
Predict Similarities	POST	/clustering/pipeline/predict	1. textToclassify: String 2. pipelineName: String

D. Abbreviations

AD – Architectural Description

ADD – Architectural Design Decisions

AKM – Architectural Knowledge Management

AMELIE - Architecture Management Enabler for Leading Industrial software

ML – Machine Learning

SC – SocioCortex

URL – Uniform Request Locator

MLlib – Machine Learning library