| | |
|---|---|
| **Grant Agreement nº** | 732463 |
| **Project Acronym:** | OpenReq |
| **Project Title:** | Intelligent Recommendation Decision Technologies for Community-Driven Requirements Engineering |
| **Call identifier:** | H2020-ICT-2016-1 |
| **Instrument:** | RIA (Research and Innovation Action |
| **Topic** | ICT-10-16 Software Technologies |
| **Start date of project** | January 1st, 2017 |
| **Duration** | 36 months |

# D2.1 OPENREQ approach for analytics & requirements intelligence


**Lead contractor:** HITEC

**Author(s):** HITEC, TUGraz, ENG, UPC



**Submission date:** October 2017

**Dissemination level:** PU



Project co-funded by the European Commission under the H2020 Programme.

**Abstract:** A brief summary of the purpose and content of the deliverable.

The goal of OpenReq is to build an intelligent recommendation and decision system for community-driven requirements engineering. The system shall recommend, prioritize, and visualize requirements. This document reports on the conceptual model of the collection, processing, aggregation, and visualization of large amounts of data related to requirements and user reviews. In addition, it reports on research results we already achieved for the purpose of WP2.

# Table of Contents

# List of Figures

# 1. Introduction

The goal of OpenReq is to build an intelligent recommendation and decision system for community-driven requirements engineering. The system shall recommend, prioritize, and visualize requirements. This document reports on the conceptual model of the collection, processing, aggregation, and visualization of large amounts of data related to requirements and user reviews.

## 1.1. Requirements Intelligence

We define requirements intelligence (RI) similarly to business intelligence (BI) but with a focus on requirements of software products. RI is about the systematic collection, analysis, processing, and visualization of requirements and user feedback coming from natural text such as issue trackers, app reviews, or legacy requirements.

The goal of RI is to provide new insights about software products based on diverse data sources of natural language and meta data. These insights can be generated from software product descriptions and used to analyze the product features and compare it to its competitors. More insights can also come from explicit feedback such as user reviews, or from implicit data gathered from the software product itself.

Further, data can come from different channels such as social media, issue trackers, emails, and other media. Combining data from different perspectives and channels leads to new ways of creating and adapting requirements as they are based on facts (e.g., implicit feedback) and customers' subjective impressions (e.g., explicit feedback).

User feedback can be used to recommend, for example, software features the customers often wishes for, which the stakeholders can then turn into requirements. OpenReq will identify those actively discussed software features and recommend them to the stakeholders. The stakeholders might agree with the discussion and turn the discussed feature to a requirement.

RI will also provide insights to understand user needs, especially for companies that are overwhelmed by the huge amount of data, by aggregating the opinion and sentiment of a large number of reviews related to their software product. In this document, we describe the technical approach to mining, processing, and visualizing requirements-related data.

## 1.2. Structure of the Document

The document is structured as follows. Section 1 gives a brief overview of the purpose of the document by introducing the first idea of a requirements intelligence unit. Then, in Section 2 we explain the state of the art related to the mining, classification, clustering of requirements and user feedback, and their visualization. Based on the results of the state of the art, we show the first version of the conceptual model in Section 3. We present an overview of the architecture related to the goal of this document and describe the components that are part of the proposed architecture. Within the components, we explain their goal, methods, the techniques we might use, and their challenges. In Section 4, we present two papers which are the result of research in OpenReq that are related to this deliverable. Section 5 summarizes the document.

# 2. State of the Art

This section has the goal to summarize the State of the Art related to the topics of this deliverable, namely the collection, processing, aggregation and visualization of large amounts of data of requirements and user feedback. Therefore, we split this section into four topics of interest. The first part will describe work related to text-mining procedures for the identification of requirements. The second part shows the state of the art in classifying requirements into functional and nonfunctional requirements, bug reports, and feature requests. Thereafter, we present work on clustering requirement documents and user reviews. The final part shows how the processed data can be visualized using diverse visualization techniques.

The state of the art focuses on *methodological* aspects—i.e., how a specific task is accomplished in the context of requirements intelligence. To that end, we leveraged our expertise to identify key references in the scientific and practitioner literature.

## 2.1. Mining Requirements

Requirements engineering traditionally involves end users to capture their needs and to get their feedback with the goal of building the right product for them. This is traditionally achieved through on-site customers' visits, interviews, focus groups, and workshops.

Requirements are initially stated in natural language which can pose problems of ambiguity, vagueness, and misinterpretation. Therefore, formalized specifications have only a small role in the specification of requirements.

A survey by *Mich et al.* [29] found that 79% of companies use unstructured natural language in their requirements documents; 16% use structured natural language (e.g., using templates); 5% use formal approaches. Therefore, the application of natural language processing (NLP) to requirements engineering has attracted a lot of attention from software engineering researchers and practitioners.

NLP is a field that stands at the crossing between computer science, artificial intelligence and computational linguistics and is concerned with programming computers to process large amount natural language text [44]. NASA built a tool that leverages NLP for requirement engineering (Automated Requirements Measurement, or ARM) already in the late 1990s [7].

The recent state of the art regarding the identification of requirements focuses on text mining, natural-language processing, and sentiment analysis [39]. Therefore, we have text mining to retrieve documents from the internet and to turn them into data. Sentiment analysis takes the processed, machine readable text from the natural-language procedure and extracts the opinion of the user. Sentiment analysis tools such as SentiStrength [61] extracts how positive and how negative a certain formulation is and relates it to a value from -5 very negative to +5 very positive.

The identification of user needs and features that satisfy such needs is an example of the current application of NLP to requirements intelligence. In fact, the features of a software system can be identified by mining the descriptions of software products and its related reviews [20], [22]. In the product descriptions, the developer or the company behind the product or service tries to explain the existing functionalities. On the other hand, in software reviews, users state what they like or dislike, explain features they think are missing in the current product, or report

bugs. Understanding the set of features of one's own software, the set of features of the competitors, and the opinion of the users, can improve the product [19], [39]. New requirements can be created or existing requirements can be updated from this information.

For example, Maalej et al. [28] show that software vendors not only look into app reviews to adapt their release cycle in case multiple people report a similar bug, but also use user feedback to get inspiration about new software features. However, users' feedback often lacks context. Therefore, usage and interaction data with the product can help requirement engineers to better understand the circumstances under which user submit feedback [59], [60].

Mining is not only applied to feedback *per se* but also to user rationale—i.e., the reasoning and justification of human decisions, opinions, and beliefs. In software engineering, rationale management focuses on capturing design and requirements decisions and on organizing and reusing project knowledge. Research is working on the identification of user rationales such as how users argue and justify their decisions [43].

Rationale management is a topic that has been researched for several decades [5], [14], [21]. One sub-topic of this line of inquiry is design rationale as discussed by Lee [25] who—through a human-computer interaction perspective—identified seven issues for design rationale systems, such as cost-effective use, domain-knowledge generation, and integration. Bruegge and Dutoit [4] and Dutoit et al. [14] describe the activities of creating, maintaining, and accessing rationale models. Further, they discuss the issues related to managing rationales in decision support and negotiation. Burge et al. [5] show how rationale can be used for decision making in the software life cycle. A summary of the state-of-the-art in rationale management can be found in Dutoit et al. [14] who shows challenges, such as the struggle to capture and maintain model bases rationale.

Specifically, in the context of requirements engineering, Kurtanović and Maalej [43] studied an automated way to mine rationale from user reviews. Researchers and tool vendors started to discuss and work on automated ways to analyze, filter, and synthesize software-related feedback into actionable advices for developers. One way to automatically mine rationale is to look at existing documentation [33]. Liang et al. [26] introduce an approach that learns design rationale from patent documents. Techniques that can be used to mine and parse information from existing documents have been analyzed by Rogers et al. [33]. The authors show that the use of linguistic features improve the precision of classifiers but significantly lowers recall compared to text mining.

Another sub-topic is argumentation mining from natural language text (e.g., legal, political, or journalistic [32]). Researcher have been mining arguments at different levels [27], [31], from the structure of the written text (e.g., premises, conclusions, and whole arguments) to the relationship between arguments. More concrete scenarios are covered by Boltužić and Šnajder [3] who analyzed reasoning in online discussions by identifying comment-argument pairs. A challenge in this field is to define what an argument actually is and how arguments can be captured automatically.

In summary, requirements engineering research is leveraging a plethora of unstructured, textual data to identify customers' needs, their sentiment towards a product, and their rationale. This is made possible through the extensive use of NLP and text mining techniques. The OpenReq Requirements Intelligence will build upon the approaches reported in this section to mine requirements.

## 2.2. Classification of Requirements, Bug Reports and Feature Requests

One topic in research is the classification of requirements into functional (FR) and non-functional (NFR) [42], [15], [35]. While there is some kind of an agreement on the definition of FRs, there is no clear definition of NFRs [16]. Both types of requirements are distinguished by whether they describe the functionality of the system (FR) or its properties and constraints (NFR) [45], [24]. This differentiation influences the practical use of requirements elicitation, documentation, and validation [12], [15]. In contrast to functional requirements, non-functional requirements are often specified later in the process and not explicitly managed [10], [11], [34], [36]. This led to a situation in which NFRs are often regarded as less important and delayed to a later point in the project, despite often influencing FRs [8], [11]. One approach to mitigate this issue is to automatically classify requirements into functional and nonfunctional. For example, Kurtanović and Maalej [42] employ a supervised machine learning algorithm that correctly classifies raw text requirements into FR and NFR 92% of the time.

Similarly, distinguishing between bug reports and feature requests is also an important topic. Recently, research has focused on app stores—platforms to distribute mobile software where users can give feedback in terms of written text or through a rating system. With more than 5 million apps and hundreds of million users' reviews, app stores are a rich source of information. By automatically mining information from app reviews, developers can leverage the users' voice and improve their app [28]. For instance, developers could check the sentiment of reviews associated with certain features to understand how users perceive them [17]. On the other hand, due to their sheer number, many reviews are non-informative and need to be filtered out [9], [30].

In summary, the application of machine learning approaches to classification problems, interesting for requirements engineering, is showing good results. The OpenReq Requirements Intelligence will implement its classification functionalities based on the approaches reported in this section. Likewise, it will address the issues, here identified, of noisy data and bulkiness.

## 2.3. Clustering Similar Text Elements and Finding Duplicates

In software projects, especially open source projects, there are often issue trackers where users can submit bug reports and feature requests. Popular projects have hundreds or even thousands of open issues. Once a new issue is submitted, one of the project maintainer needs to check and mark duplicates. Such task is important as it helps to reduce the number of open issues to those that matter. Research has tackled this problem by introducing automated approaches for detecting duplicates [40], [18].

In addition, research had tried to understand whether duplicates are harmful or not. For example, Bettenburg et al. [2] found that duplicates are created because previous reports lack information and that duplicates can add value by including more information. Sometimes, bug reports are only interesting for developers if they have a certain level of severity and if a critical mass of people are affected [28]. Therefore, there are approaches that group issues by their type (e.g. bug reports, feature requests) and then cluster them by their similarity (e.g., based on NLP metrics, such as tf-idf). These clusters need to reach a minimum size to be considered for the next release of the software [37].

The specific approach to issues and bug report clustering varies a lot based on context. For example, developers and technical savvy stakeholders attach the stack trace (i.e., information about the active subroutines of the program affected by the bug) to the report. Consequently, machine learning techniques leverage these information to group stack traces together, which are more structured than natural language text, and extrapolate similar reports [53]. Evaluated internally at Microsoft, this approach resulted in a F-measure of 0.88 and can facilitate diagnosis and prioritization of issues to be addressed. However, as the authors report [53], its efficacy in large scale open source projects.

Similarly, Anvik et al. [54] augmented natural language processing features with execution information about the context in which the issue/bug was observed. They were able, training their model on the Firefox bug report dataset, to detect up to 93% of duplicate bugs (compared to the 72% of using natural language features alone).

Clustering issue trackers items and bug reports according to their text element is useful to automatically generate summaries. To that end, Rastkar et al. [55] clustered bug reports leveraging their conversational features[1] such as the position of the sentence in the comment, the author of the comments and her involvement in the discussion (e.g., number of comments per author), the normalized length of the sentences, and their lexical features [56]. Their approach outperformed two similar clustering models built using data from a different domain (i.e., email threads and email threads with meeting minutes) in terms of precision and recall and yielded a F-score of 0.40. Human validation showed that the results are coherent and non-redundant [55]. Having a summary of the existing issues, bugs or feature requests simplifies the identification of duplicates as the stakeholder (e.g., software developer, product owner, or requirements engineer) needs to consider less textual information to make a decision.

Other approaches leverage the latest advancement in information retrieval (IR) to group together similar bug reports and find duplicates. Sun et al. [57] built discriminative models based on 54 textual features trained with three large open source datasets (i.e., Eclipse, OpenOffice, and Firefox bug reports). The authors' approach shows a ~15% improvement in recall rate with respect to findings of [54] showed above.

In summary, a mix of NLP and information retrieval techniques are being used to tackle clustering problems in the context of requirements engineering feedback. The OpenReq Requirements Intelligence will build its relevant functionalities (e.g., finding duplicate feedback) on these approaches while taking into account the identified challenges, such as context diversity.

## 2.4. Visualizing Requirements

Visualizations of large amounts of data, related to requirements and user feedback, augment the ability of the resulting requirements artifacts to reach a wide range of stakeholders and provide for a rapid and shared understanding of complex information.

There is the need for visualization techniques that can serve to enhance communication and understanding relative to the essential properties from which a software systems will be developed.

[1] Bug reporting systems allow developers to comment on a bug creating an actual conversation between developers

Abad et al. [62] emphasize on the importance of the visualization in requirements engineering and present a systematic literature review (SLR) in which the authors analyzed 26 primary papers. In their work, the authors summarize the analyzed papers by presenting proposed visualization techniques to RE related dimensions. The three dimensions are RE activities (incl. elicitation, modelling, communication, verification, and evolving), stakeholders (incl. end users, developers, decision-makers, and customers), and finally, domain (problem- and solution domain). In a later step, the authors relate the proposed visualization techniques to their visualization functions. These functions are e.g., coordination, attention, and motivation. Eventually, six visualization types such as diagrams and sketches are mapped to the papers that used them to support RE related activities.

Cooper Jr. et al. [58] present a survey on the state of the art in visualization in requirements engineering. They tackle the issue of effective visualization by surveying 29 research papers presented in Requirements Engineering Visualization (REV) workshops from 2006 to 2008. In their work, they relate visualizations to 1) the lifecycle of RE including context and groundwork, structured specification, evolution, and maintenance and 2) to RE activities along the lifecycle such as elicitation, verification, and validation. During their analysis, they found the following five categories of visualizations:

- *Tabular visualizations*. Tabular visualizations are made up of a series of intersecting rows and columns.
- *Relational visualizations*. Relational visualizations consist of a collection of nodes and connectors that describe or indicate a relationship between components or a system, but do not implicitly describe the inherent order of operation of the system. For example:
    - ❖ *Network Graph*: this visualization supports undirected and directed graph structures. This type of visualization describes relationships between entities. Entities are displayed as round nodes and lines show the relationships between them. The vivid display of network nodes can highlight non-trivial data discrepancies that may be otherwise be overlooked (Figure 1).

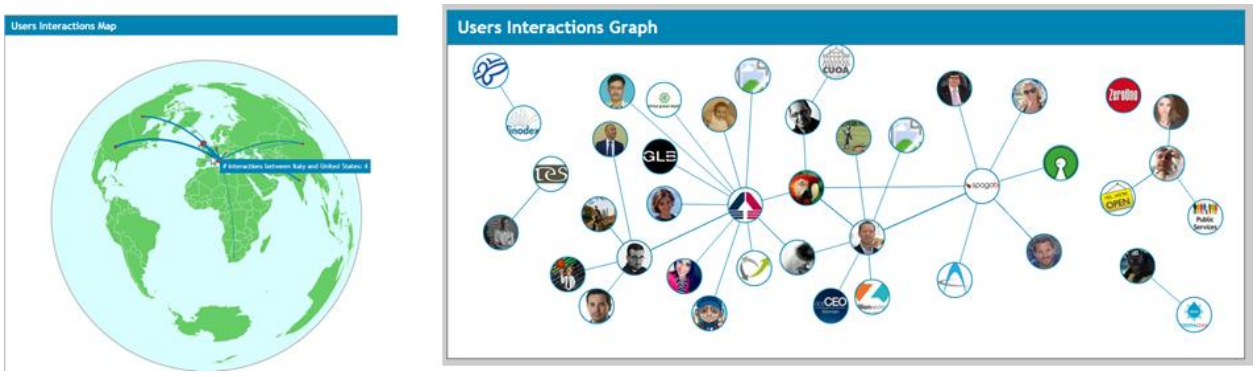


**Figure 1: Examples of Network graph visualization.**

- *Hierarchical visualizations*. Hierarchical visualizations imply the decomposition of a system and its parts, as typically used in goal-based modeling approaches.

  For example:

- *Sunburst*: This type of visualization shows hierarchy through a series of rings, that are sliced for each category node. Each ring corresponds to a level in the hierarchy, with the central circle representing the root node and the hierarchy moving outwards from it.

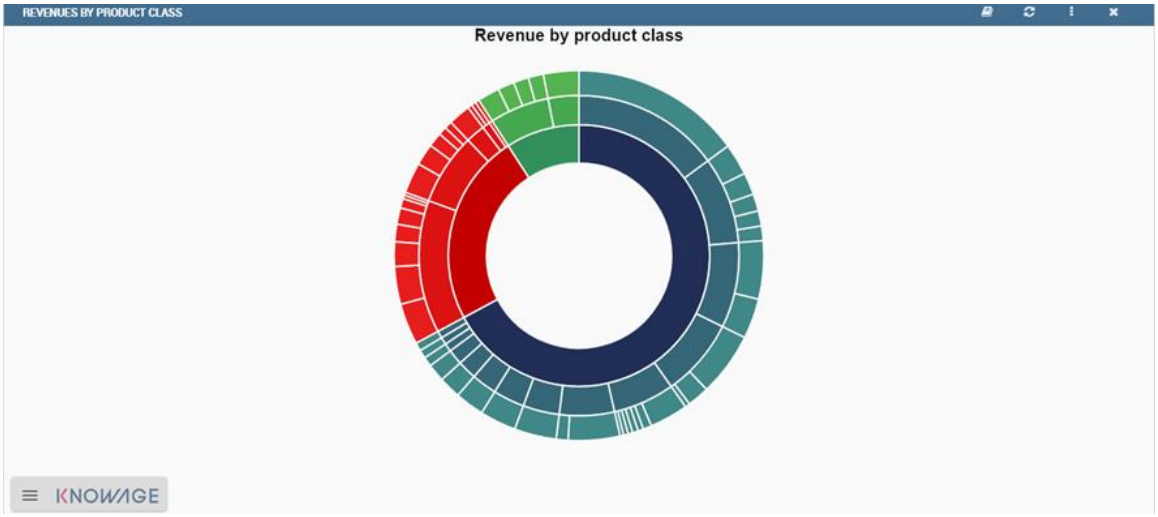


**Figure 2: Examples of Sunburst visualization.**

- *Treemap*: display hierarchical data as a set of nested rectangles. You can drill down in the data, and the theoretical number of levels is almost limitless. You can use a treemap when space is constrained and you have a large amount of hierarchical data that you need to get an overview of. Treemaps should primarily be used with values that can be aggregated.

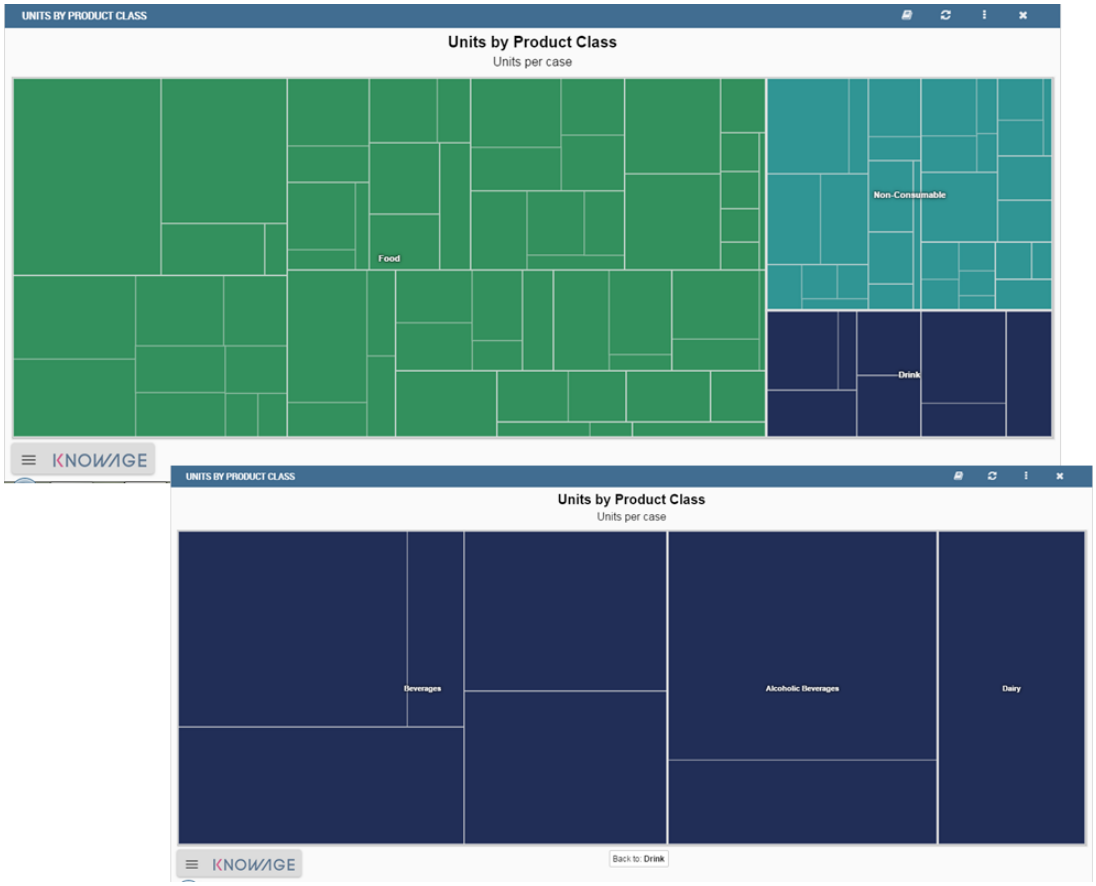


**Figure 3: Example of Treemap visualization.**

- *Quantitative/Metaphorical visualizations*. Quantitative visualizations are most commonly seen in the form of bar graphs, pie charts, line graphs, or other figures that convey relative data, but also include more sophisticated techniques that make use of visual metaphors and other visual clues such as color, shape, line thickness, and size to convey meaning at a glance.

  For example, there are the following graphs:

  - *Chord:* a graphical method of displaying the inter-relationships between data in a matrix. The data is arranged radially around a circle with the relationships between the points typically drawn as arcs connecting the data together. This kind of graph could be used to represent a distribution of the number of messages respect a category and a subcategory.

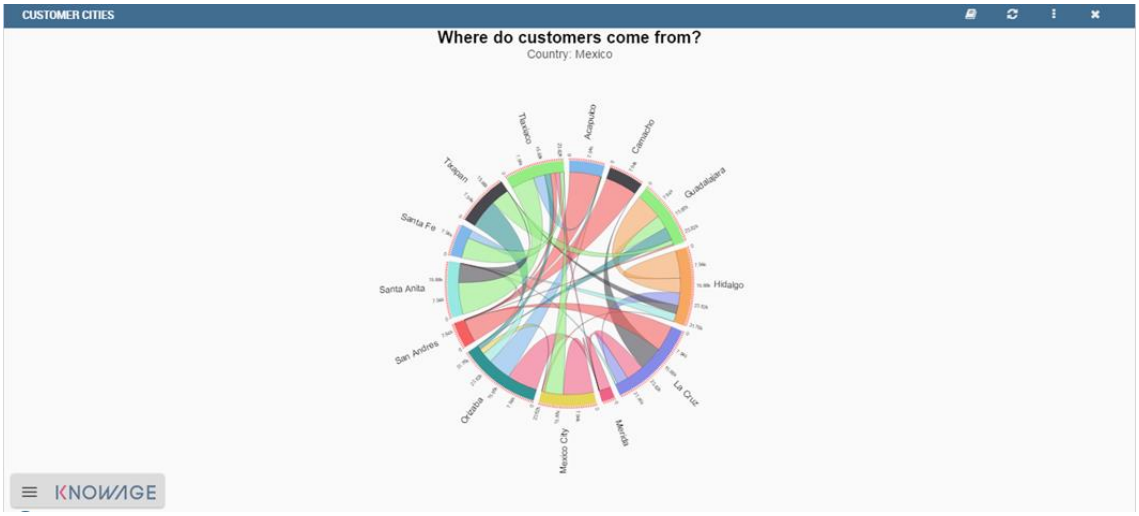


**Figure 4: Example of Chord visualization.**

- *Gauge visualizations:* use a radial scale to display the data range and a dial to indicate the data value, like a speedometer. A gauge is used to display a specific data point using a dial over a radial scale with defined limits. Colors can be associated for the sections of the data to suit the application e.g. green for satisfactory, yellow for caution, and red for alarm. This kind of graph could be used to represent a sentiment in feedback analysis.

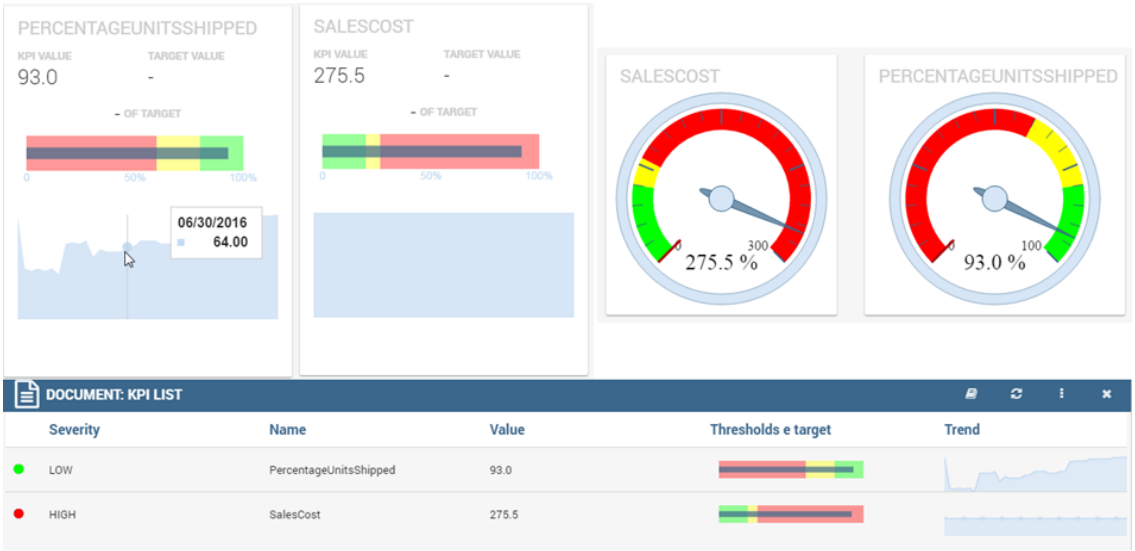


**Figure 5: Example of Gauge visualization.**

- *Heatmap:* is a graphical representation of data where the individual values contained in a matrix are represented as colors. A heat map chart can be used to visualize complex data like performance comparison of different companies, market response, stock market investments, and lots more. There can be many

ways to display heat maps, but they all share one thing in common—they use color to communicate relationships between data values that would be much harder to understand if presented numerically in a spreadsheet. This kind of graph could be used to represent for example the number of explicit feedback for topic and period.

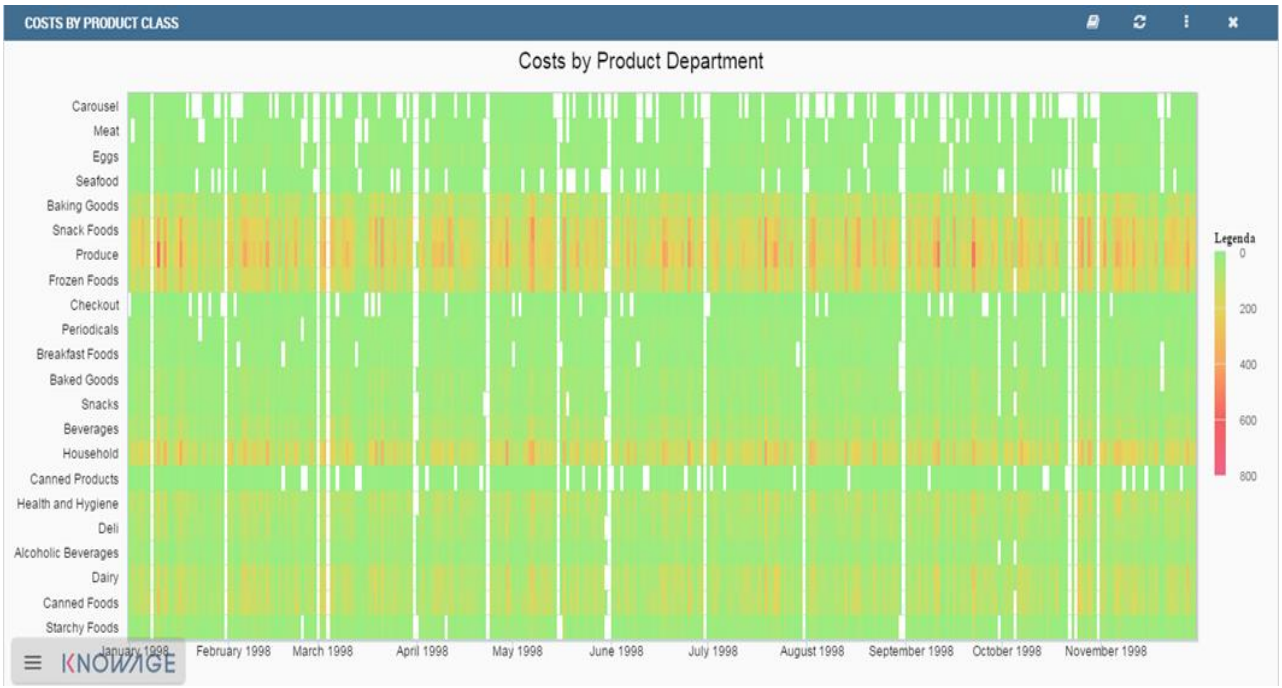


**Figure 6: Example of Heatmap visualization (costs by product and month).**

- *Radar:* displays multivariate data in the form of a two-dimensional chart of three or more quantitative variables represented on axes starting from the same point. For example, we can use a Sentiment Radar as in the Figure 7 to visualize the sentiment distribution for different topics.

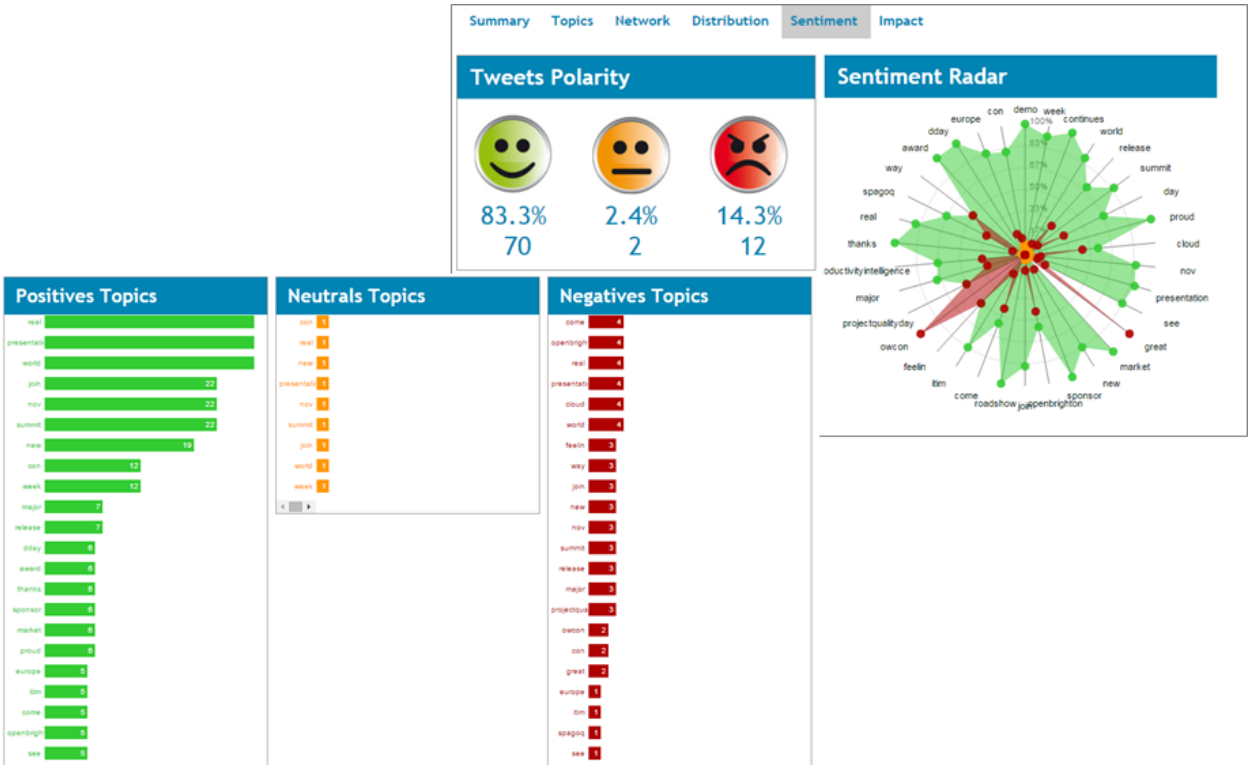


**Figure 7: Example of Radar visualization.**

- *Scatter:* a type of plot or mathematical diagram using Cartesian coordinates to display values typically for two variables. If the points are color-coded, one additional variable can be displayed. The data is displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis. This plot could be use in a clustering and classification analysis.

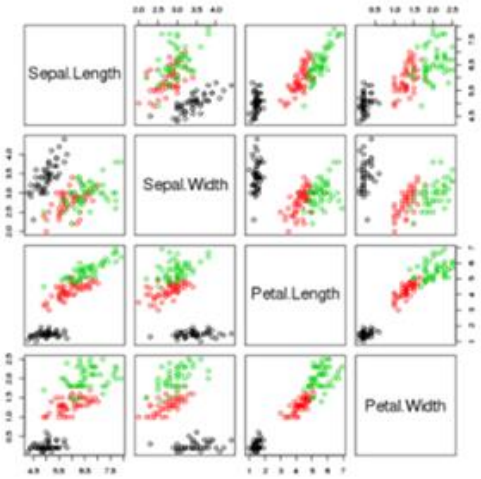


**Figure 8: Example of Scatterplot visualization.**

- *Word Cloud*: visual representation of text data, the importance of each tag is shown with font size or color (for example we can use it for Hashtag Cloud and Topics Cloud)

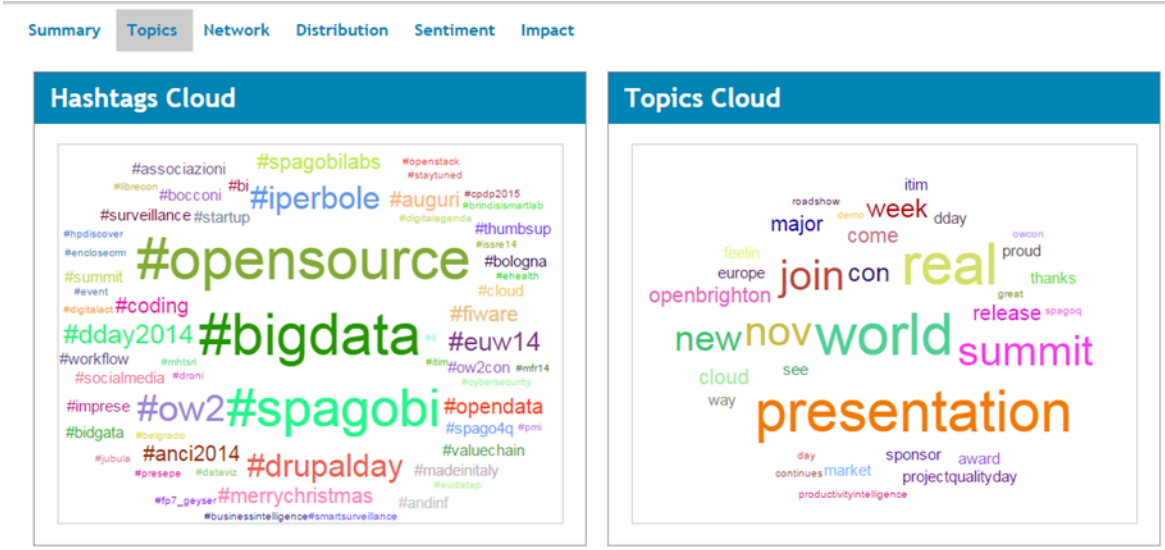


**Figure 9: Example of Word cloud visualization.**

In the context of OpenReq, we can combine different analysis, using graphs and tabular data, in a single dashboard. Moreover, an OLAP Analysis is possible applying the classical operation of:

- *Pivoting*: to rotate the data axes to view the data from different perspectives. For example, cities could be arranged vertically and products horizontally while viewing data for a particular quarter. Pivoting could replace products with time periods to see data *across* time for a single product.

- *Slicing:* to take out the slice of a cube, given certain set of select dimension (i.e. customer segment), and value (i.e., home furnishings) and measures (i.e., sales revenue, sales units) or KPIs (i.e., Sales Productivity). A rectangular subset of a cube is selected by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension.
- *Dicing:* produces a sub cube by allowing the analyst to pick specific values of multiple dimensions.
- *Drill-down:* to navigate among levels of data ranging from the most summarized (up) to the most detailed (down).
- *Roll-up:* A roll-up involves summarizing the data along a dimension hierarchy.
- *Drill-through:* A Drill Through capability allows users to view relational transactions that make up a multidimensional point in an OLAP Cube. The Drill-Through therefore brings to light data in the Cube constituted from the Data Source, which often is within an RDBMS.

This kind of tool could be used, for example, to analyze the implicit feedback.

# 3. RI Intelligence Architecture

This section presents the conceptualization of the approach to RI that will be followed in OpenReq and the main requirements that drove the choices for designing the conceptual model of the OpenReq RI Engine. Following such requirements, an overview of the architecture is presented together with a description of its several components.

## 3.1. RI Infrastructure Requirements

The requirements for the infrastructure are driven by two main constraints: high scalability and data flexibility.

The RI Engine is to be designed with special attention to scalability as a large amount of data will be collected from different sources of stakeholders (e.g., users). Considering scalability using two dimensions, the x-axis deals with running multiple copies of the application across servers, whereas the functionalities that are available scale over the y-axis. The former is addressed using load-balancing mechanisms which is nowadays a requirement for IaaS providers (e.g., Amazon Web Services, Microsoft Azure). This scalability dimension is usually addressed at a hardware level (i.e., through specialized servers and networking equipment) and is, therefore, not discussed in this document. The second level of scalability drove the decision to select a microservice architecture. Following such approach, the RI Engine functionalities are broken down into components and services so that each loosely-coupled service is responsible for a one or few closely related functionalities.

At the same time, the RI Engine leverages a diversified set of data sources, most of which are unstructured (i.e., natural language). The approach deals with the need for two diverse set of data representation with a hybrid approach to data and metadata management. Therefore, on top of a relational storage mechanism which imposes a rigid data representation (i.e., a schema), the RI Engine uses a database model that does not separate the data from its schema. The relational storage is used for operations on data that has a strictly defined structure that is unlikely to change (e.g., bug report entry); the non-relational solution is used for data that cannot be constrained in shape as it changes from one instance to the other (i.e., the representation of a machine learning model).

The two solutions presented above (i.e., microservice architecture and hybrid storage) address the *volume* (i.e., large amount of data in terms of size), *velocity* (i.e., frequency of incoming data to be processed), and *variety* (i.e., unstructured data) requirements typical of modern analytics platforms.

The requirement about high scalability impact the overall architecture of the RI Engine (see Section 3.2), whereas the requirement regarding data representation impacts the *DSL* of the proposed architecture (see Section 3.3.12).

## 3.2. Architecture Overview

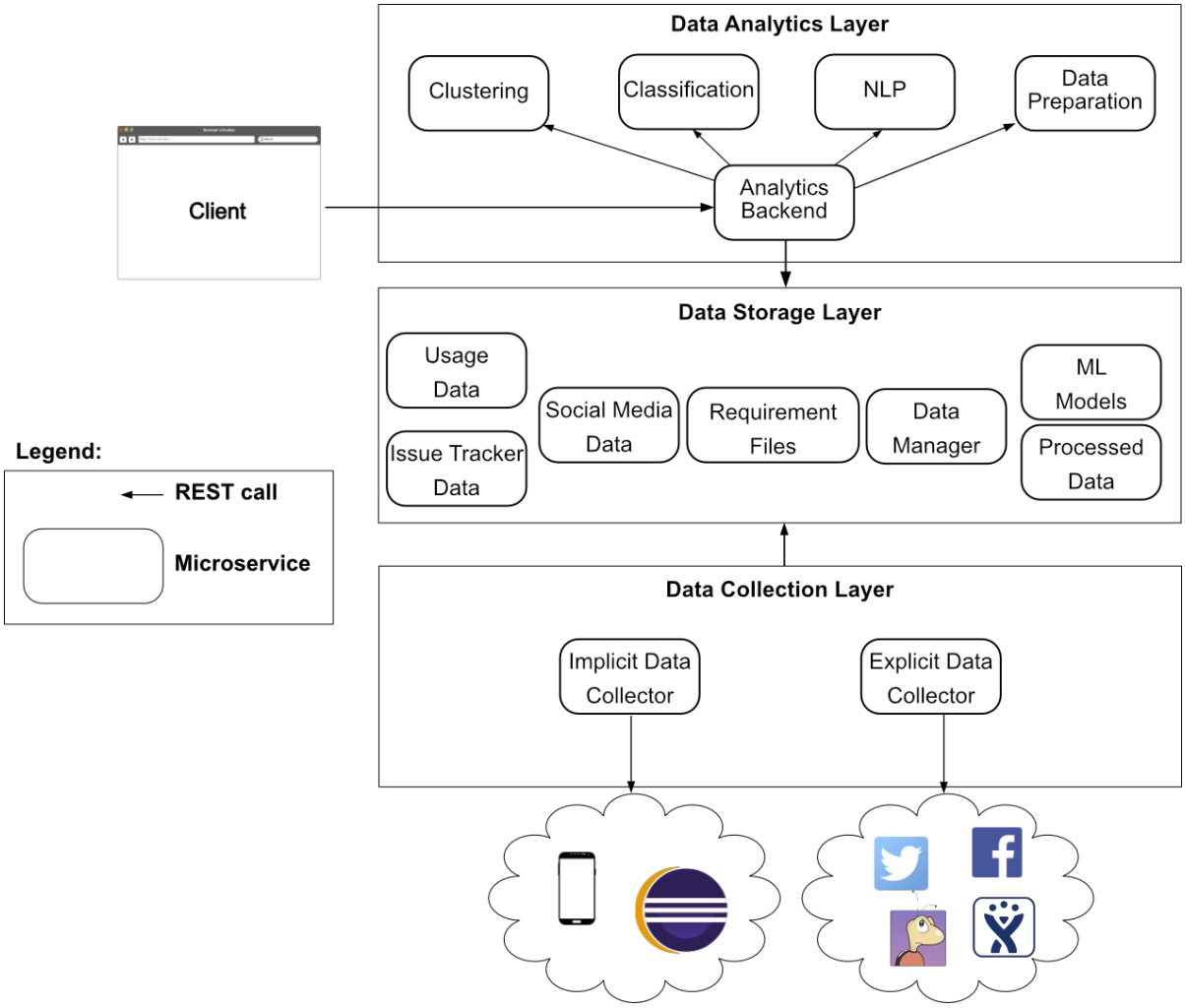


**Figure 10: Architecture overview of the requirements intelligence unit.**

In OpenReq, we emphasize a microservice (MS) architecture (Figure 10) because this architectural style provides us the following advantages:

1. We can develop each MS in the programming language the expert team feels most comfortable with.
2. This style allows us to scale because each MS can run on its own machine or even create duplicates on several machines.
3. MS are highly decoupled software components with a focus on small tasks, which enables us to easily exchange each MS as long as we follow their designed API.
4. MS requires a strong and detailed API.
5. Microservices are highly reusable as they are self-contained and usually have well-documented APIs.
6. Maintaining MS can be performed by the related expert team and does not require knowledge about other microservices but only the APIs that needs to be satisfied.

On the other hand, this architecture introduces, among others, overhead due to the orchestration of the several MS (e.g., increased effort in deployment, monitoring, and service discovery) and their compatibility (e.g., keeping dependent services compatible when updating a single service).

For a better visualization, we grouped the microservices into three layers: data analytics (DAL), data storage (DSL), and data collection (DCL). In the following, we discuss each layer in a separate section. Each of these sections contain all of its related microservices.

## 3.3. Data Analytics Layer (DAL)

This layer is the interface to the frontend of OpenReq and moderates all processing steps of natural language, as well as log and context data. The goal of this layer is to provide all requested information in a human readable way.

The microservices in this layer mainly use open-source components for machine learning and natural language processing. In the following we briefly present prominent examples of existing open-source components.

Scikit-learn[2] (BSD license) offers a Python implementation of the most popular machine learning algorithms for classification, clustering, and regression. The library is characterized by an excellent documentation, high performance, and ease of use. It is well maintained (~22,000 commits from more than 800 contributors) and is de-facto industry standard for machine learning with Python. In Java, Weka[3] (GNU General Public License) is a popular choice for data mining. Weka includes a graphical user interface to get results with few clicks to load and analyse data. It supports diverse machine learning approaches for classification, clustering and attribute selection. In addition to the graphical user interface it can be accessed via the command line or the Java API.

Nltk[4] (Apache 2.0 license) and Gensim[5] (LGPL license) are used for natural language processing tasks. The former performs common tasks, such as tagging, tokenizing, and stemming; the latter is designed to be efficient with large, unstructured texts and implements algorithms that facilitate the extraction of recurring patterns of words in the set of documents. MALLET[6] (CPL 1.0 license) presents a Java implementation of similar algorithms, however it is less maintained and documented. Stanford CoreNLP[7] (GNU General Public License v3+) is an often used NLP toolkit for Java, it supports NLP tasks such as named entity recognition, dependency analysis, and part-of-speech tagging. This toolkit currently has 79 contributors and is still maintained by the Stanford NLP group.

### 3.3.1. MS: Clustering

The overall goal of this microservice is to group similar texts. The MS can be used on any domain that uses natural language such as issue tracker comments, social media posts, or app

[2] http://scikit-learn.org/
[3] http://www.cs.waikato.ac.nz/ml/weka/
[4] http://www.nltk.org/
[5] https://radimrehurek.com/gensim/
[6] http://mallet.cs.umass.edu/
[7] https://stanfordnlp.github.io/CoreNLP/

reviews. It enables us to, for example, find issues in issue trackers about the same topic, helping us to find duplicates. For instance, Villarroel et al. [37] use the clustering algorithm DBSCAN [50] to find similar bug reports that were mentioned by different users in app reviews.

The microservice can use different clustering algorithms like k-means, DBSCAN, and HDBSCAN [50]. The strength of k-mean is that it can easily handle large data but the number of clusters (i.e., categories of issues in a issue tracker) must be defined a priori. Using HDBSCAN, there is no need to specify the number of clusters explicitly. However, it does not scale well.

A challenge in clustering text is to understand the domain and the input as most clustering algorithms depend on manually setting an a priori number of clusters for a given set of documents. HDBSCAN is an approach that finds the number of clusters on its own, however, experimentation with data coming from different sources is needed.

### 3.3.2. MS: Classification

Classification is concerned with identifying a class/category of an unseen observation. Classifiers might solve a binary classification problem (i.e., two available classes) or a multi-class problem.

This MS encapsulates the classification of processed text elements, its related metadata, and, on a higher level, a requirement itself [42], [43]. Classifiers can be used to, for example, identify bug reports and feature requests in issue tracker, comments, or app reviews [28]. They can also be trained to identify informative vs. uninformative comments [9]. Specifically for OpenReq, we will use classifiers to automatically distinguish between functional and non-functional requirements, predicting rationales, identify bug reports and feature requests, and filter out uninformative issues/reviews.

The challenges in classification problems deal with achieving high precision and recall, which are needed to embed the classifier in a tool. Precision represents how many observations were correctly classified. Recall represents how many correct observations are returned by the classifier. The F1-score (i.e., the harmonic mean of precision and the recall) is used to present results embedding both metrics. Extensive experiments are necessary to understand what data is needed and how classifier parameters should be tuned for an accurate classification in terms of F1-score.

### 3.3.3. MS: NLP

Natural Language Processing (NLP) is a process that enables computers to understand and manipulate natural language. We envision to use several NLP techniques such as stopword removal, stemming, lemmatization, tense detection, and bigrams detection. Therefore, the objective of this MS it to extract *what* stakeholders are talking about and *how* they perceive specific functionality from text. In the following we describe some of the techniques we will use.

Stopwords are common English words such as “the”, “am”, and “their”, which typically have a grammatical function and do not influence the semantic of a sentence [46]. Research has shown that removing them from text can reduce noise. Lemmatization [46] is the process of reducing different inflected forms of a word to their basic lemma to be analyzed as a single item. For instance, “fixing”, “fixed”, and “fixes” become “fix.” Similarly, stemming reduces each term to its basic form by removing its postfix. While lemmatization takes the linguistic

context of the term into consideration and uses dictionaries, stemmers just operate on single words and therefore cannot distinguish between words which have different meanings depending on the part of speech they appear in. For instance, lemmatization recognizes “good” as the lemma of “better” and stemmer will reduce “goods” and “good” to the same term.

Both lemmatization and stemming can help the classifier to unify keywords with the same meaning but different language forms [17]. For instance, a classifier will better learn from the reviews “crash*ed* when I opened the pdf” and “the new version crash*es* all time” as the term “crash” is an indication for a bug report. Finally, we can also use bigrams of the n-gram family. Bigrams are all combinations of two contiguous words in a sentence. If we have the sentence: “The app crashes often,” the bigrams are the following: “The, app”, “app, crashes”, and “crashes, often”. If used by the document classifier instead of single terms, bigrams allow to additionally capture the context of the word in the review [20]. The identification of the sentiment within a sentence or a user comment can be helpful to get a deeper understanding of the users’ opinion [47]. By extracting the part of speech (POS) of a sentence and by employing a grammatical analysis, we are able to get information regarding the software features users mention in comments [22].

The main challenge with NLP in our context is language ambiguity. Language ambiguity makes processing texts difficult as there are words with multiple meanings (e.g. bank) and synonyms (e.g. take photo and capture photo probably means the same). Further, the use of the language differs based on the data source (e.g., comments in issue trackers, tweets, app reviews, formal software product descriptions) and the people who write the text (e.g., users, analysts, developers). For instance, some reported issues might follow strict guidelines and best practices that allow users to clearly state the problem, the steps to reproduce the issue, and additional information (e.g., code snippets or screenshots), while others might just state that some kind of problem exists without giving much details. Moreover, people use language differently, as some prefer a colloquial style, while others are more formal in their writing.

An issue, specific to the OpenReq project, is the diversity of languages used to write the text analyzed by the NLP component. The majority of the existing NLP approaches (e.g., stemming, lemmatization, sentiment analysis) target the English language as they are trained and validated using English text corpora. However, for the specific case of the telecom trial, the component will deal with text written in the Italian language and, for the case of Siemens, with text (partially) written in the German language. Although, NLP approaches and software libraries exist for both languages [48], [49], their performances (e.g., precision) might be inferior compared to the well-established, English-based ones.

### 3.3.4. MS: Data Preparation

The data preparation microservice is responsible for preparing the data for all following steps. For the natural language processing steps, text data must be separated from meta-data (e.g., timestamp, format, star rating). If the input data is e.g., html, all html related tags and content must be removed, leaving only the information we are interested in. Other documents that we want to analyze with NLP might not need to be passed to this microservice such as written requirements documents.

If we collect explicit data, we are most likely to retrieve html documents that the crawlers got from the target websites. Html data is verbose as it contains i.e., tags, JavaScript, and imports. In order to clean that data, we need to extract the information, we are interested in by means

of html parsers such as jsoup[8] or beautiful soup[9]. Some websites provide APIs that return (most of the time) json[10] data. Json data can also be interpreted by the data preparation microservice and extracts the fields necessary for further computation.

The challenge of this microservice is to provide preparation steps for multiple different data sources (implicit vs. explicit feedback) and to handle different data types (html, json, interaction data).

### 3.3.5. MS: Analytics Backend

This microservice is the access point for the OpenReq client (e.g., a web browser). For each request, the data analytics backend has a configured pipeline that decides which of the other microservices of this layer has to be called, in which order, and with what configuration. For instance, there is a case in which we want to extrapolate all bug reports reported on Twitter. This example has the following four steps:

1. Get the Twitter dataset from the *DSL*.
2. Call the data preparation microservice to separate meta-information from the tweet text.
3. Use the NLP microservice to transform the text.
4. Call the classification microservice to classify all tweets.

Before a pipeline is called, the analytics backend checks if the processing was already performed. If the data is already processed, it then gets loaded from the *DSL* to reduce unnecessary computation.

The challenge is to create and evaluate a well-defined set of pipelines for the requests that will be available through the OpenReq platform.

## 3.4. Data Storage Layer (DSL)

All microservices in this layer have the same goal, summarized in this subsection (see Figure 10). In this layer, we want to persist the data coming from the data collectors but also want to store the intermediate results of the DAL for easier, faster reuse—e.g., avoid training a machine learning model every time we need it.

Each MS in this layer may use different technologies depending on the data they manage. Some may use relational data representations (e.g., MySQL[11], GPL 2.0 license), some might use less structured, document-oriented representations (e.g., MongoDB[12], AGPL 3.0 license), and others might use the file system. An initial overview of how the data is organized is presented in Figure 11.

[8] https://jsoup.org/
[9] https://www.crummy.com/software/BeautifulSoup/
[10] http://www.json.org/json-de.html
[11] https://www.mysql.com/
[12] https://www.mongodb.com/

### 3.4.1. MS: Usage, Issue Tracker, Social Media & Requirement Files

Those simple microservices provide data, according to their specific data sources, to the services in the *DAL*. The *Usage Data MS* stores data collected by the *Implicit Data Collector* (see Section 3.3.14), whereas the *Issue Tracker*, *Social Media*, and *Requirement File* store data collected from the *Explicit Data Collector* (see Section 3.3.15). From a data perspective, these microservices use separate entities (i.e., Usage, Issue, Social Media, and Requirements File in Figure 11) to store and retrieve data (i.e., content and metadata).

The MS encompass basic read/write functionalities, whereas the complexity of dealing with the heterogeneous data is deferred to the underlying MS in the *DCL* (see Section 3.3.13). Therefore, they do not pose any substantial challenges.

### 3.4.2. MS: Data Manager

This microservice orchestrates tasks related to the storage and retrieval of data. On one hand, it routes data from the *DCL* to the appropriate microservice (see Section 3.3.6). On the other, it stores the abstracted data (i.e., content and metadata) used by the machine learning algorithms in the *DAL*. From a data model perspective (see Figure 11), the *Feedback* entity represents such abstracted data containing both content (i.e., raw text) and metadata (i.e., user's rating, syntactic information)

The Data Manager helps decoupling the raw data from the models data. One challenge is to update the Data Manager in case a new data source, together with its MS in the *DCL*, is added to the architecture. Similarly, the Data Manager needs to be updated once a new type of machine learning model needs to be persisted.

### 3.4.3. MS: ML Models

This microservice acts as a caching mechanism to avoid re-computing machine learning models in the *DAL*. From a data perspective (see Figure 11), it stores the models result in the appropriate entity—*Classification*, *Cluster*, or *ML Model*. The *Classification* entity stores data related to classification operations on the *Feedback* (e.g., feedback classification as non-functional requirement for a feature request expressed with a negative sentiment); accordingly, *Cluster* will store information regarding automatically identified groups of feedback (e.g., feedback concerning the same topic). Finally, *ML Model* stores the results of a generic machine learning model (e.g., pattern-based similarity) that operates on clusters or classes of feedback rather than on single ones.

The main challenge posed by decoupling the actual machine learning algorithm from its storage mechanism (for either its input, parameters, and results) is to assess an appropriate level of abstraction. Thus, the changes necessary to synchronize this microservice with the one implementing the algorithm itself, once a new one is added to the *DAL*, should be minimal. Such flexibility can be achieved by using data storage that supports semi-structured data.

### 3.4.4. MS: Processed Data

Through this MS, we store the machine learning features for training the machine learning models extracted from metadata, natural text, and context data. This avoids, together with the *ML Models MS* (see Section 3.3.11), re-computing the entire input when new data is crawled/received. Therefore, only the new data points need to be processed. This approach brings flexibility in creating and testing new models in case the parameters or the algorithm

itself needs to change. From a data perspective (see Figure 11), this MS stores and provides the *Parameters* (e.g., features, context, metrics) for the ML Models. Such approach decouples the input data relative to the OpenReq data sources from the model specific input.

This microservice depends on the type of machine learning models used in the *Data Analytics Layer*. Therefore, one challenge is to make sure that the *Processed Data MS* supports the type of data that a new machine learning model requires. This challenge is addressed by using a flexible storage solution based on semi-structured data.

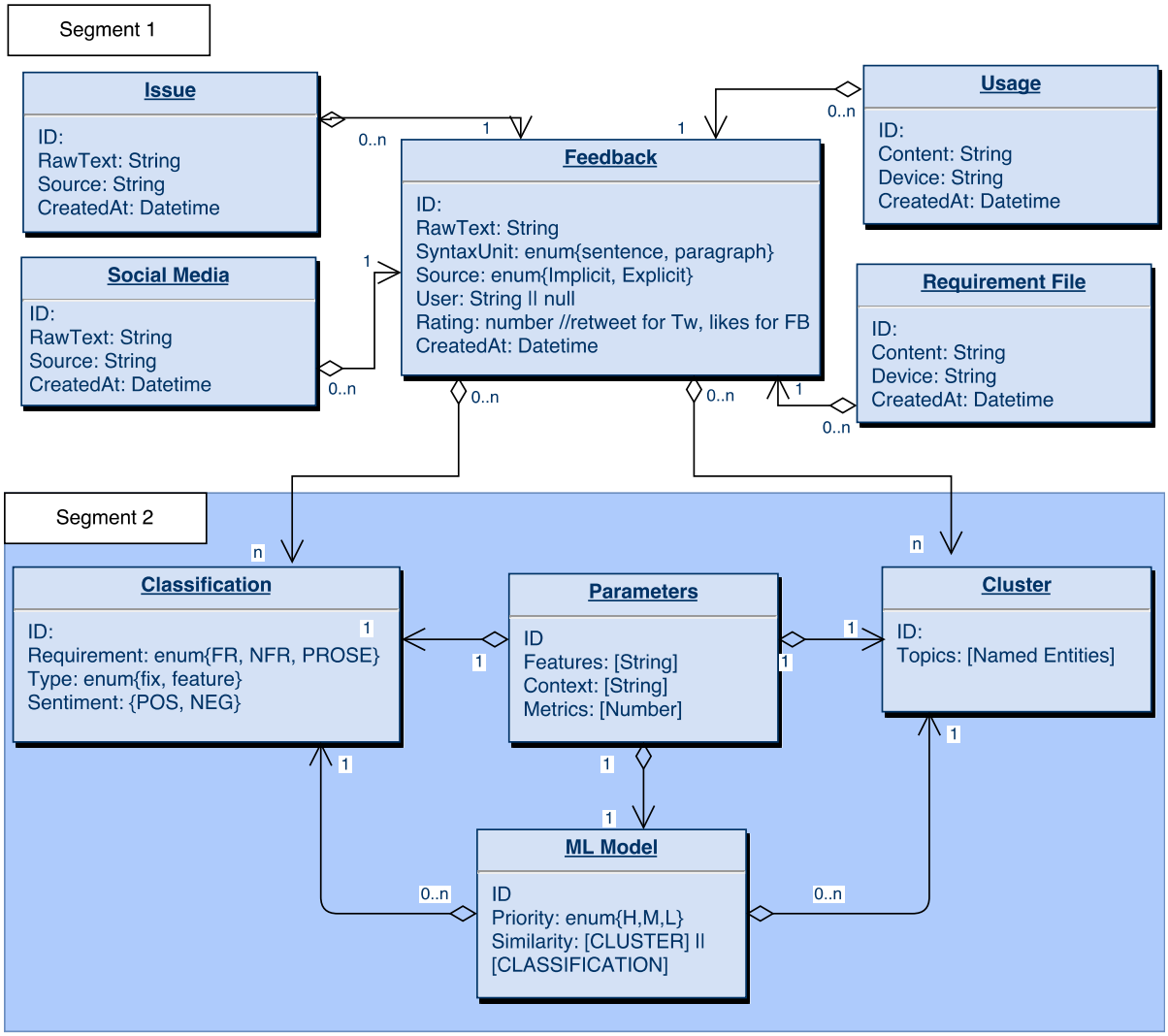


**Figure 11: Data entity relationship model of the Data Storage Layer.**
(Segment 1 includes the entities operating on the OpenReq data sources in the Data Collection Layer, whereas Segment 2 the includes entities operating on the machine learning models from the Data Analytics Layer)

## 3.5. Data Collection Layer (DCL)

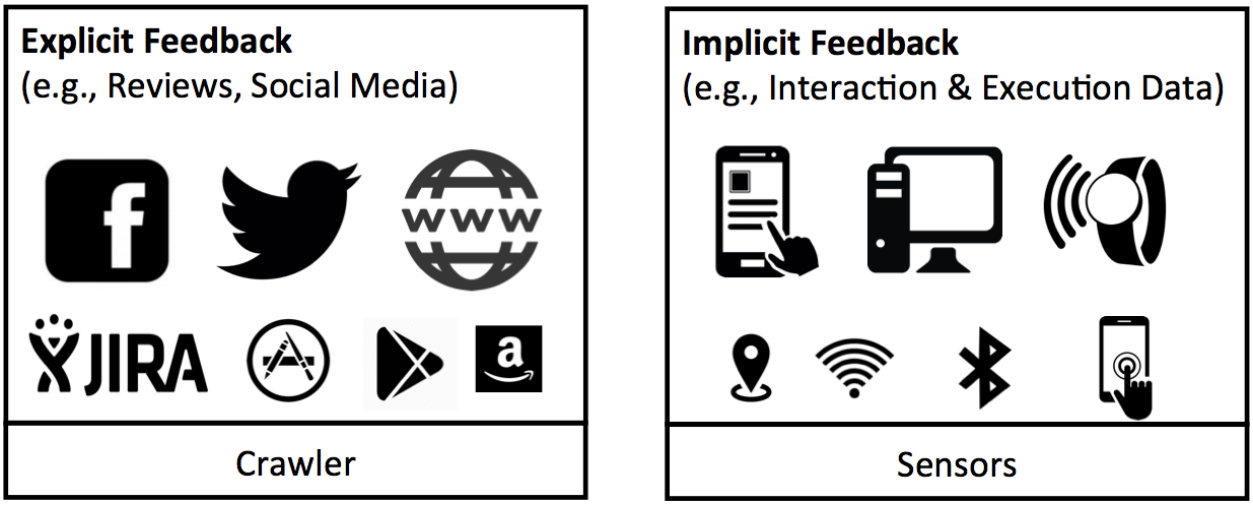


**Figure 12: Comparing explicit with implicit feedback by showing possible data sources.**

As shown in Figure 12, OPENREQ will collect two types of user feedback: explicit and implicit feedback. Explicit feedback is collected by means of crawlers, while implicit feedback needs to be collected with a software library that contains sensors.

The collection of the explicit feedback is implemented using the API of the data source when available (e.g., Twitter[13]). Otherwise, ad hoc crawlers will be implemented using open source web-scraping framework, such as Scrapy[14] (BSD license), a Python library for automatically extracting structured data to be later processed and stored, or BeautifulSoup[15] (MIT license) which allows a more low-level control over the parsing of HTML or XML documents.

The collection of the implicit feedback requires the instrumenting of the device with an ad hoc software (e.g., a mobile app). Such software is developed ad hoc for the platform of interest (e.g., Android, iOS).

Besides these techniques, we can make use of already existing open source projects such as the Horizon 2020 project SUPERSEDE (project id: 644018). SUPERSEDEs goal to improve users' quality of experience. For this they propose a feedback-driven approach for the life cycle management. The project uses diverse sensors for Android devices and web applications. For instance, the Component "FAME" is responsible for the acquisition and monitoring of user feedback[16]. In addition, the European FP7-ICT project MUSES (project id: 318508) is also an open source project that provides source code for a context data collection framework on Android and iOS devices. The goal of MUSES was to improve corporate security by analyzing context and interaction data on different devices to warn users whenever they violate

[13] https://dev.twitter.com/rest/public
[14] https://scrapy.org/
[15] https://www.crummy.com/software/BeautifulSoup/
[16] https://github.com/supersede-project/monitor_feedback

corporate policies. In OpenReq we will analyze existing approaches to look for reusable components that fits the purpose of OpenReq.

### 3.5.1. MS: Implicit Data Collector

Implicit feedback is a strong source of information in OpenReq compared to explicit feedback where people have to state their opinion *after* they used, for example, a tool. Implicit feedback on the other hand collects the interactions of the users *during* the usage. Collecting interaction data is an important source of information as it represents the true behavior of a user in a certain context, such as working on an issue in the Eclipse IDE. The information gathered through this channel is objective, as we do not collect emotions or opinions of users.

Implicit feedback can be collected by deploying software on a device of the target group. This can be a plugin in the Eclipse IDE or an app on a smartphone that collects time-stamped context data. We define context as *data that describes the users' and the devices state*. We categorize this data as task, physical, and computing environment context. The task context describes what users are doing on their devices, like sending an email or more generally doing some work. The physical context represents information about the user's physical environment, such as location and time. The computing environment includes information that describes the device itself—e.g., operating system, installed applications, and network connections.

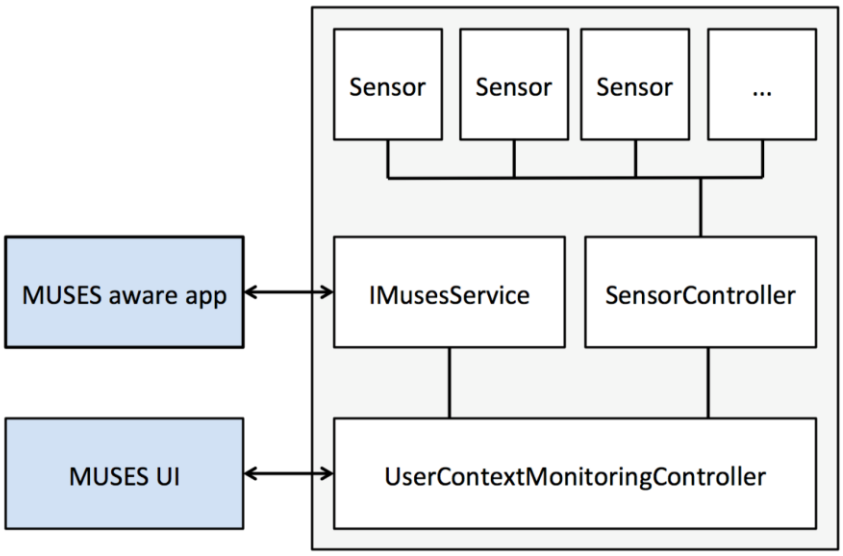


**Figure 13: Sensor based architecture from the European FP7 project MUSES project to collect implicit feedback.**

Figure 13 shows a conceptional architecture for collecting context information (implicit feedback) on mobile devices. This architecture was used and tested in the European FP7 project *MUSES* to collect context data in real-time to enforce security policies [52]. The architecture allows to add different sensors, depending on what information should be collected and what is allowed to be collected from a privacy perspective. It has a central unit called *UserContextMonitoringController* which enables the application to not just collect data but to also communicate the results internally and to third-party applications. Therefore, users can decide what kind of data they are willing to share.

The main challenge, from a legal perspective, is to find how much data can we collect from the user in a way that it is proportional to the use [51]. Being transparent with what data will be collected and giving the user the choice to allow and disallow some data collection can be a good way to handle legal issues.

A second challenge is represented by usage data that cannot be directly accessed (e.g., software cannot be deployed on the user's device). To overcome such limitation, the context can be recreated, at least partially, from proxy infrastructure usage data (e.g., interactions between the base station antenna and the device).

### 3.5.2. MS: Explicit Data Collector

The explicit data collector's goal is to actively look for user comments on the world wide web. Possible data sources are social media posts on Twitter or Facebook, issues in issue trackers, or user reviews of software products in app stores. We also consider requirements presented in a more structured format as an additional data source. Accordingly, we will collect data from plain-text requirement documents (e.g., Microsoft Word, Microsoft Excel, PDF) and machine-readable files (e.g., ReqIF[17]).

Depending on the data source, we either call their publicly available API or crawl their website manually by developing a customized software once no API is available. Requirements in a textual format will be manually imported from their storage (e.g., filesystem, requirements management tool). Explicit feedback is able to capture emotions and opinions which would not be retrievable by relying only on implicit data.

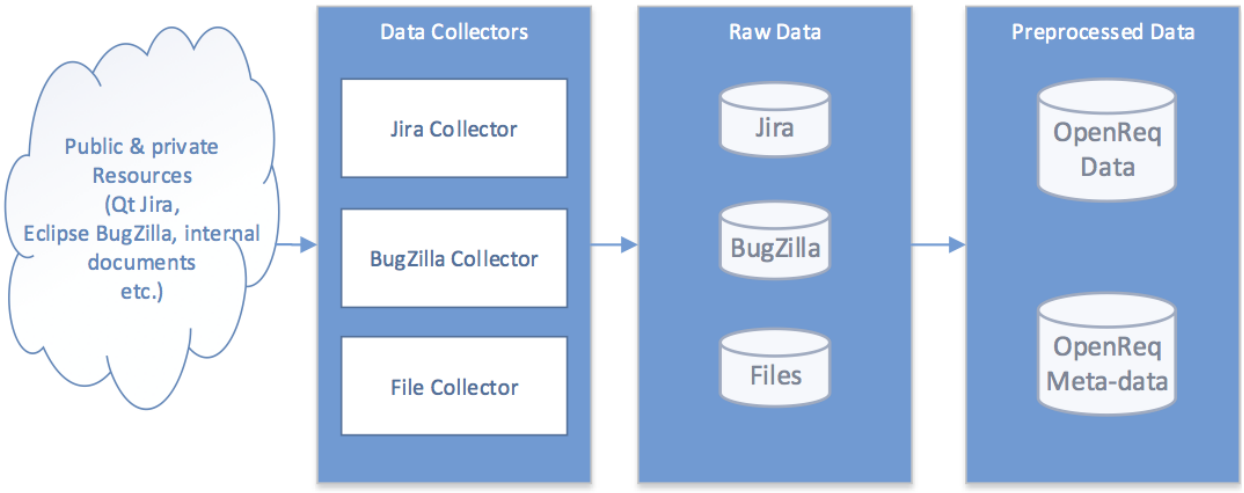


**Figure 14: The approach to collect explicit feedback.**

[17] http://www.omg.org/spec/ReqIF/

Figure 14 shows the explicit data collection approach of OpenReq. Specialized data collectors[18]—implemented in the *Explicit Data Collector* MS (see Figure 10)—will collect data from different sources and store them into data sets through the appropriate MS in the *DSL* (see Section 3.3.10).

The challenge of this microservice is to develop and maintain a crawler for each data source as the structure of the data and the number of data sources itself might change over time.

## 3.6. Components for Data Visualization

In this section, we describe two open source software components (i.e., Knowage and Birt) that can be used to visualize the data and analytics provided by the RI Engine, as described in Section 2.4. We have evaluated also other open source reporting tools (JasperReports, Pentaho) but they have licenses incompatible with the EPL license. Indeed, the Impact Committee was directed to this kind of license and this decision can limit our choices. Only Birt has an EPL license, so this component does not generate issues for license reasons. Nevertheless, though the license of Knowage is not compatible with the EPL too, we consider this platform could be useful and used for particular analysis.

Knowage is indeed a complete and very rich platform for business analytics, derived from the long history of SpagoBI. It has inside not only a reporting component but also much more. Furthermore, we have all the competencies to make the most of it, because Knowage was developed by Eng itself.

We can bypass the license issue, if needed, with the only condition that, when we distribute the OpenReq software, we will indicate the link from which Knowage software can be downloaded and how to integrate it with the rest of OpenReq during installation. Knowage contains also BIRT inside and it is much richer of functionalities as default.

So, we could use both the components, according to the needs.

### 3.6.1. Knowage

Knowage[19] is the open source business analytics suite of Engineering that combines traditional data and big data sources into valuable and meaningful information. It merges the innovation coming from the community with the experience and practices of enterprise-level solutions.

The suite is composed of several modules, each one conceived for a specific analytical domain. They can be used individually as complete solution for a certain task, or combined with one another to ensure full coverage of user' requirements, allowing to build a tailored product.

The components interesting for the requirement intelligence context are:

- *Big Data (BD)*: combine structured historical enterprise data with external multi-structured ones. *Knowage BD* gives the opportunity to work with big data sources and traditional ones, federating data sets to build different analysis, such as static reports, maps, network views, interactive cockpits and data/text mining models.

[18] We use the terms "collector" and "crawler" as synonyms.

[19] https://www.knowage-suite.com/

Moreover, the user can freely explore his own data using a drag & drop query builder or having immediate insights thanks to advanced visualizations.

- *Smart Intelligence (SI)*: usability, ad-hoc reporting, free inquiring, self-management for the end-user. *Knowage SI* provides the opportunity to work with traditional data sources, even combining more, to build analysis such as interactive cockpits, reports and multidimensional analysis (e.g., OLAP). The product supports the IT staff managing enterprise environment with multi-tenant and complex metadata. At the same time, it enables the end users to freely inquiring the own data space and producing self-made analysis and visualizations.
- *Predictive Analysis (PA)*: makes hypothesis and evaluate their impacts in an interactive way. Predictive Analysis concerns the ability to perform advanced processing using data mining techniques for forecasting and prescriptive purposes. Moreover, it means being able to simulate actions and evaluate their effects on different assets. *Knowage PA* gives the opportunity to work with traditional data sources or external files, processing them with advanced algorithmics written using R/Spark and other scripting languages. For what-if purposes, Knowage PA provides an OLAP-based solution that allows the interactive simulation process over measures and dimensions.
- *Embedded Intelligence (EI)* makes Knowage open. It can be linked to third-party solutions under the terms and conditions of the AGPL v3 license. *Knowage EI* gives the opportunity to work with Java APIs, Javascript APIs, REST services and some other technical interfaces to provide SSO support or the integration with external user repositories.

Through the Data Mining engine and the Knowage Function Catalog, is possible to run R and Python scripts interactively and visualize their graphic output. Is thus possible to integrate custom algorithms in Knowage and release outputs or functions to the end user.

From a technical standpoint, Knowage is a set of web applications that can be installed on any Java Enterprise Edition compliant application server, such as Apache Tomcat, RedHat JBoss Application Server, IBM Webpshere, or Oracle WebLogic.

The Knowage repository, which stores all the objects that govern its operations, as well as any analytical documents developed, can be hosted on any database for which a JDBC driver is available, such as Oracle, MySql, PostgreSQL, Sql Server, etc.

Knowage has been designed to be configured in a cluster environment to meet the high reliability and scalability requirements of the solution. It can be installed on multiple nodes in a server cluster to get high reliability and continuity of service, or define multiple instances of the same Knowage application in order to deploy distribute workload (e.g., for complex reports it will be executed by a dedicated instance). In addition to managing the horizontal application server's scalability, Knowage has an architecture that allows you to manage scalability flexibly in front of increased volume or data features to manage.

Knowage manages the concept of *datasource* as an object that encapsulates the connection characteristics to a data source, such as:

- *traditional data sources* (e.g., Oracle RDBMS, MySql, PostgreSql, MS Sql Server) or any sources that allow access via JDBC standard.

- *non-traditional data sources* (e.g., Big Data, NoSql) using specific mechanisms for each connection. For example, MongoDB—a document-based open source database for which it is not necessary to define an a priori schema—does not provide a JDBC driver. However, a specific Java connector implemented by Knowage allows access to MongoDB data.

A key concept in Knowage is the *dataset* that allows decoupling data retrieval modes from presentation modes. It has two interfaces:

- A Connector-type interface that encapsulates data source querying (i.e., an SQL query, a script, a REST service).
- An Adapter interface that encapsulates the way the data contained in the Data Set and read through the Connector is made available to the analysis and display tools (e.g., columns of a dataset, XPath).

The following diagram explains in detail the concept of Knowage DataSet:

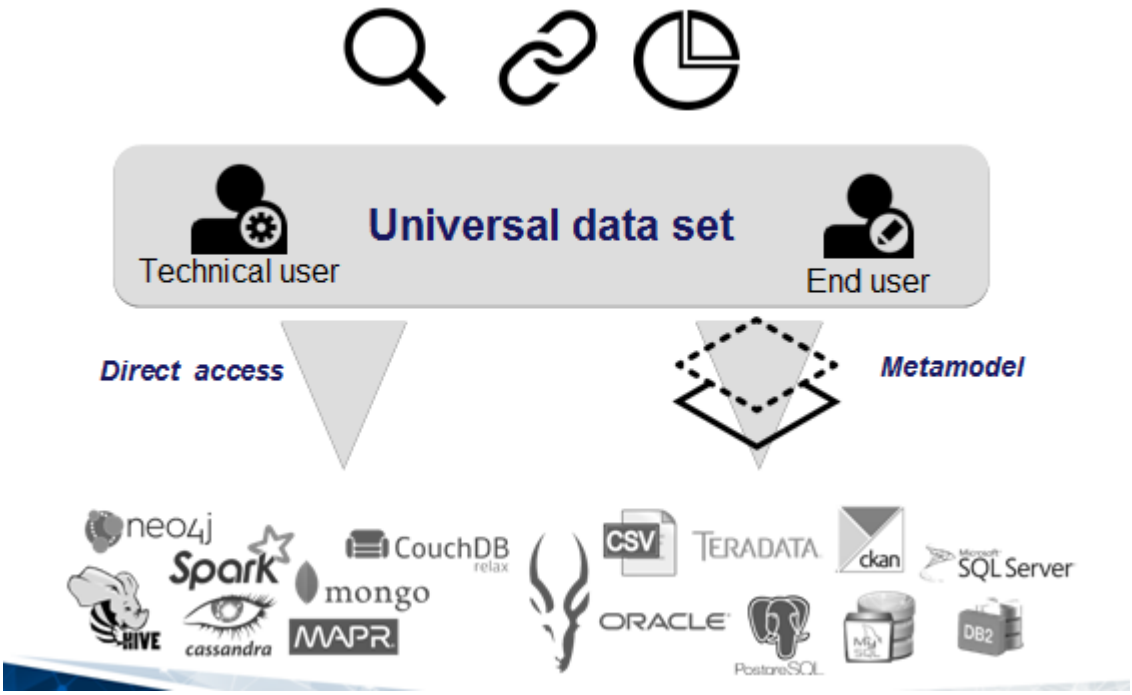


**Figure 15: The Knowage dataset.**

A meta-model is an abstraction of the database that makes any data source easily manipulated by the end user without having to know query or programming languages.

Knowage can be integrated with external systems through APIs or using an SDKs.

With regard to security, the behavioral model of Knowage uniquely defines the concepts to be applied for safe content management (which features are available with respect to user roles, which data the user has permission to read, etc.).

Knowage is available in two versions:

- Knowage Community Edition (CE), with an open source license (AGPL licence with constraints), ensuring full utilization of analytics functionalities and full operativeness of the final user.
- Knowage Enterprise Edition (EE), with a subscription model, facilitating management operations for the installed park and ensuring the professional services required for use in business settings.

### 3.6.2. Birt

BIRT[20] is an open source software project that provides the BIRT technology platform to create data visualizations and reports that can be embedded into rich client and web applications, especially those based on Java and Java EE.

BIRT has two main components: a report designer based on Eclipse, and a runtime component that can be added to an application server. BIRT also offers a charting engine that lets the users add charts to their own application. BIRT designs are persisted as XML and can access a number of different data sources including JDO datastores, JFire Scripting Objects, POJOs, SQL databases, Web Services and XML.

The reports available are:

- Lists - The simplest reports are lists of data. As the lists get longer, the users can add grouping to organize related data together (orders grouped by customer, products grouped by supplier). For numeric data, totals, averages and other summaries can be added.
- Charts - Numeric data is easier to understand when presented as a chart. BIRT provides, among others, pie charts, line & bar charts. BIRT charts can be rendered in SVG and support events to allow user interaction.

The BIRT project delivers many components. These are listed below with a brief description.

- BIRT Report Designer - an Eclipse perspective is used to create BIRT report designs. These designs are stored in an open XML format. The Designer can be downloaded as a Rich Client Platform (RCP) application, a set of plug-ins to enable the Designer perspective within an existing Eclipse build or as an all-in-one Eclipse instance.
- Design Engine - this engine is responsible for creating and modifying report designs. The Design Engine API (DE API) wraps the functionality of the design engine and is available for use within any Java/Java EE project. The BIRT Report Designer uses this API internally to construct the XML designs.
- Crosstabs - Crosstabs (also called a cross-tabulation or matrix) shows data in two dimensions, for instance, sales per quarter or hits per web page.
- Letters & Documents - Notices, form letters, and other textual documents are easy to create with BIRT. Documents can include text, formatting, lists, charts, etc.
- Compound Reports - Many reports need to combine the above into a single document. For example, a customer statement may list the information for the customer, provide text about current promotions, and provide side-by-side lists of payments and charges.

[20] http://www.eclipse.org/birt/about/

A financial report may include disclaimers, charts, tables all with extensive formatting that matches corporate color schemes.

BIRT reports consist of four main parts: data, data transforms, business logic and presentation.

- Data - Databases, web services, Java objects all can supply data to your BIRT report. BIRT provides JDBC, XML, Web Services, and Flat File support, as well as support for using code to get at other sources of data. BIRT use of the Open Data Access (ODA) framework allows anyone to build new UI and runtime support for any kind of tabular data. Further, a single report can include data from any number of data sources. BIRT also supplies a feature that allows several data sources to be combined using inner and outer joins.
- Data Transforms - Reports present data sorted, summarized, filtered and grouped to fit the user's needs. While databases can do some of this work, BIRT must do it for "simple" data sources such as flat files or Java objects. BIRT allows sophisticated operations such as grouping on sums, percentages of overall totals and more.
- Business Logic - Real-world data is seldom structured for a report. Many reports require business-specific logic to convert raw data into information useful for the user. The logic can be scripted for a single report or, in case the application already contains the logic, using existing Java code.
- Presentation - Once the data is ready, there is a wide range of options for presenting it to the user. A single data set can be visualized in multiple ways and a single report can present data from multiple data sets.

BIRT Components (architectural overview):

- Report Engine - The Report Engine uses the report design files to generate and render reports. Using the Report Engine API (RE API) the engine can be embedded within any Java/Java EE application. The BIRT Web Viewer uses this API to execute and display reports.
- Charting Engine - The Charting Engine is used to design and generate Charts either in standalone or embedded within BIRT reports. The Charting Engine API (CE API) allows Java/Java EE developers to add charting capabilities to their applications. The Design and Report Engines make use of this API to deliver Charts.
- BIRT Viewer - The BIRT project provides a sample viewer that is used to preview reports within Eclipse. In addition to being packaged as an Eclipse Plug-in, the Viewer is also available as a standalone Java EE application, which can be used in any JSP-compliant Java EE server. The Viewer Plug-in can also be embedded within a Rich Client Platform (RCP) application. BIRT provides web output as a single HTML document, paginated HTML, PDF, XLS, DOC, PPT, and Postscript.

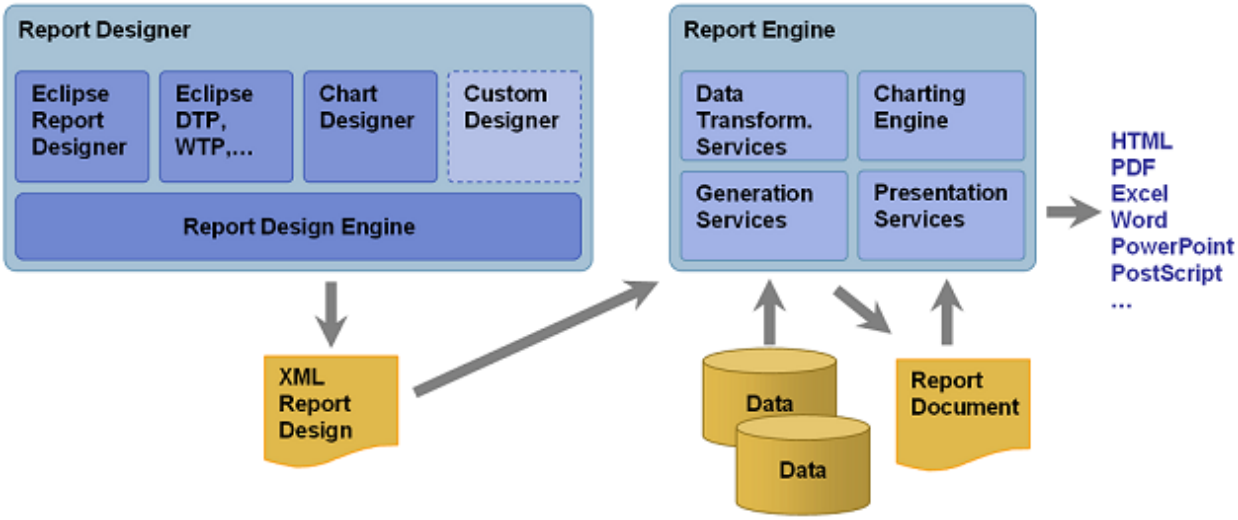


**Figure 16: BIRT components and architectural overview.**

The spectrum of reporting applications is enormous, and the BIRT team can never provide every feature needed by every application. BIRT scripting support is one way to extend BIRT. Another is to create BIRT extensions that plug into BIRT. The project provides many extension points that can be used to extend BIRT. Some of the more common ones are listed below.

- BIRT uses the Data Tools Open Data Access (ODA) framework for adding custom data access methods. Data access extensions include a runtime component for getting the data. They can also include custom design-time UI for building a custom query. For example, a packaged application vendor can use ODA to build data access UI that works with the vendor's own data model.
- BIRT provides a solid set of report items for presenting data. Applications that have specific needs can create additional report items that work within the designer and engine just like BIRT own report items. For example, a performance management application might add report items that display stop lights, gauges and other visual indications of performance metrics.
- The BIRT chart package provides a wide variety of chart types. However, some industries have developed very specific chart formats. Developers can create chart plug-ins that add these chart types into the BIRT charting engine.
- BIRT provides output in HTML, Paginated HTML, PDF, XLS, DOC, PPT, ODS, ODP, ODT, and Postscript. Many other types of output are possible: ERich Text Format (RTF), Scalable Vector Graphic (SVG), images, and more. While BIRT will add some of these over time, others may have a more limited audience. Developers can use BIRT engine interfaces to add additional converters, including those specific to a given application.

BIRT is a top-level software project within the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors and an open source community. BIRT is licensed under the Eclipse Public License (EPL).

# 4. RI Intelligence Approach

In this section, we present two papers—published by the OpenReq team—related to the requirements intelligence unit. Both papers had been accepted and presented at the 25th IEEE International Requirements Engineering Conference (RE2017). We focus on the approaches stated in the papers and explain how they can be used in OpenReq.

## 4.1. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews

This section summarizes the first paper, reporting a novel approach for extracting new requirements for mobile apps using textual data and metadata.

### 4.1.1. Motivation and Goal

Nowadays, the two most prominent app stores, the Google Play Store and the Apple App Store, contain more than three million apps and hundreds of millions of app reviews. App Stores became an interesting source of information for software developers and requirements engineers because they can potentially reach billions of people and aggregate information from customers, business, and technical perspectives. Each app is represented by an app page that contains information like the description of the app, user reviews, screenshots, and the average rating.

The goal of SAFE [22] is to provide a uniform approach that can extract app features, such as “send email” and “synchronize calendars” from two different sources; the app description and the app reviews. App descriptions are written by developers and are used for describing the functionality of the app they provide. User reviews on the other hand are the customers of the developers that may provide feedback in terms of a star rating and a written text. SAFE extracts the app features of the description and the reviews, without prior machine learning training, to analyze what features the developer provide and to understand how their users talk about it. The feature extraction outperforms two state of the art approaches by achieving an average f-measure of 46% for app descriptions and 35% for app reviews.

The final step of SAFE is a matching of features stated in the description with those users actually discuss in app reviews. This matching provides new insights to the app/software. App features that are described but not mentioned in the reviews, might be an unpopular feature or a feature people are satisfied with. We could get more information about the concrete meaning by also including context data (implicit feedback) in the future. If the app feature is not described but frequently mentioned in the app reviews a might be a feature request. If a feature is described and discussed it might be popular feature, a bug report, or a change request. The matching achieved an accuracy of 87%.

### 4.1.2. Approach

This section briefly explains the overall approach of SAFE. A summary of all steps is showed in Figure 17.

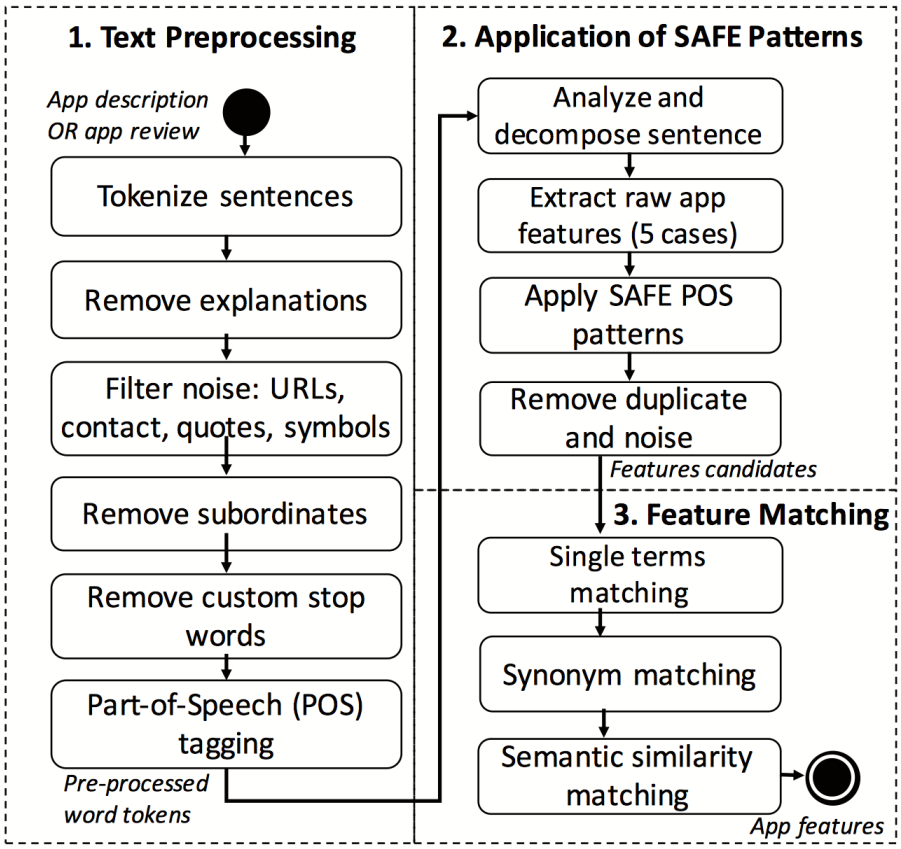


**Figure 17: The main steps of the SAFE approach.**

The input of SAFE is either an app description or an app review. First, the approach performs basic text preprocessing steps that are similar to most of the state of the art approaches. Steps like tokenization, stop-words removal, and part of speech (PoS) tagging are such common steps. In this work, we included additional steps like removing subordinates which we found helpful because they do not explain what the user is doing but rather how. After preprocessing the input, the second step is to extract the app features. To that end, we identified 18 common PoS patterns such as verb + noun (i.e., "send email", "write message") that are frequently used to describe app features. Additionally, we found five sentence structures that contain app features. One of these sentence structures is "send and receive attachments" which translates into the two app features "send attachments" and "receive attachments".

After we applied the PoS patterns and the sentence structures on the preprocessed text we can continue to match app features of the descriptions and the reviews. For the matching, we used a three-step process. In the first step, SAFE checks if each single term of the app features from both sources are identical. In the second step, we use WordNet (an ontology of the English language) to compare the synonyms of each word of the app feature. This step is important to tackle language ambiguity; people use different words to describe the same thing as "take photo" or "capture image." In the final step, we extract the semantic similarity of the app features and calculate the cosine similarity to understand how closely the features of the app description and the app review are related.

### 4.1.3. Usage Scenarios

The usage scenarios of SAFE are manifold. Perhaps the most intuitive scenario for the app feature extraction is monitoring app features health, which is to continuously identify and measure which app features are mentioned in user comments, how frequent, and their associated sentiments. In addition, app store analytics approaches that classify user reviews into bug reports and feature requests or into informative and non-informative can benefit from SAFE. If, for example, a bug report gets enriched by the addressed app feature, the information can help developers to faster trace and solve the bug.

Another scenario SAFE provides is the identification of the features delta and new insights. This can be done by identifying the differences between features described by developers and those described in the user reviews. If the delta is identified, one might want to extract further details by distinguishing between different reviews subpopulations into specific regions, stores, or channels like a forum or social media comments. Those could be recommended to stakeholder to turn them into actual requirements.

In the paper, we address the different use of language. Developers write their text more formal and often try to emphasize the app features. Users on the other hand often write colloquial, use abbreviations, emoticons, and put strong emotions in their text. Developers and users might learn how to communicate better and bridge their "vocabulary gap". On the other hand, developers can easily get the list of suggested features by users, which is a useful information for requirements identification, scoping, and prioritization. Current research classifies user reviews in e.g., feature requests and bug reports. But research often simply classifies the complete app review but does not filter the relevant information. With SAFE we might be able to also enable the clustering and aggregation of artifacts (such as comments) based on features.

## 4.2. Mining User Rationale from Software Reviews

This section summarizes the second paper, reporting the results of a study in which machine learning is used to extract rationale from textual information in user-generated software reviews.

### 4.2.1. Motivation and Goal

In paper [43], we introduce user rationale for software engineering. Since nowadays, social media, user forums, and app stores have a huge number of users, software vendors started to give these channels an increasing attention. Software vendors especially look at the input of users in order to better make decisions about software design, development, and evolution. As of this situation, this work focuses on the main questions, *is there a user rationale which can be valuable for software and requirements engineering and thus should be captured and managed?* Design rationale focuses on design related decision, user rationale on the other hand, focuses on concepts like the issues users encounter, alternatives they consider, their criteria of assessments, and their decisions. In this work, we found that issues, alternatives, criteria, and decisions often co-occur in user comments and that in 21% to 70% they contain justifications. Further, we found that the more co-occurrences exist and the longer the user comment is, the more it is likely to have a high justification density. The goal of this work is to automatically mine the concepts of user rationales from user comments.

### 4.2.2. Approach

We built a supervised machine learning approach using text, meta data, sentiments, and syntactic features using three classification algorithms (i.e., Naïve Bayes, Support Vector Machine, and Logistic Regression) along with different configurations to predict user rationale at comment and sentence level. The precision and recall for all five user rationale concepts were ranging, from comments and sentences respectively, between 80%-99% and 69%-98%. In order to achieve this result, we performed the steps depicted in Figure 18.

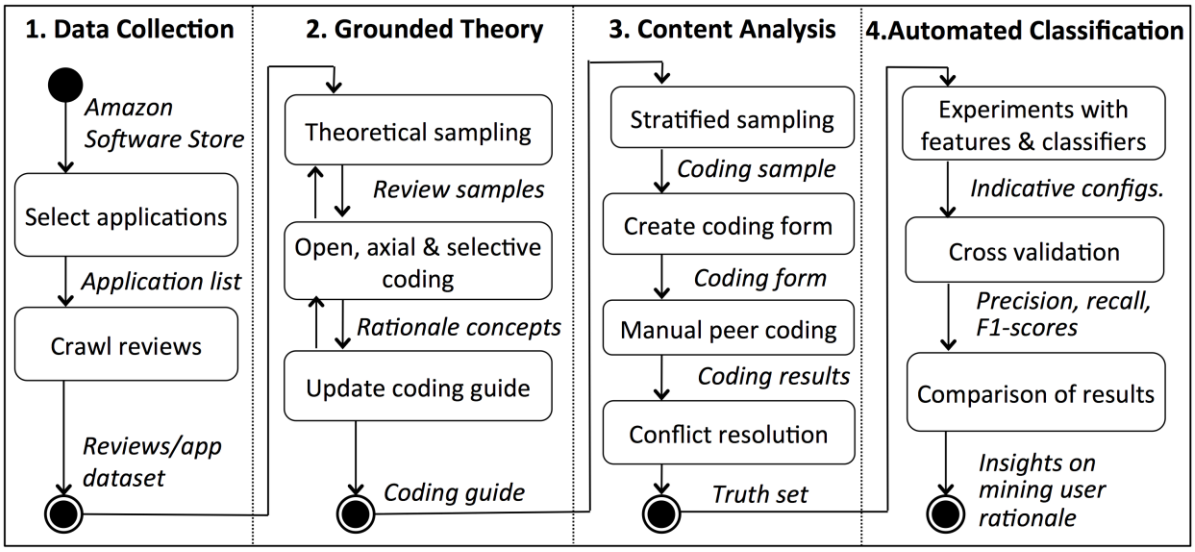


**Figure 18: Overview of our research methodology with four phases.**

First, we performed a data collection step to gather data from the Amazon software store. Similar to app stores, users can give a star rating and written feedback which includes additional metadata such as the name of the reviewer and whether the reviewer bought the product. In the second step, we followed the grounded theory approach presented in Strauss and Corbin [63] to manually analyze user reviews. We used this method because it is a systematic and qualitative way to develop a theory based on evidence in data. The goal of this phase is to create a coding guide that describes the theory that was used for the creation of the truth dataset later used to train the machine learning models. In the third step, we performed a content analysis. In this phase, we created a stratified sample of 1020 reviews, which were later peer-coded using the coding guide. The final step is the automated classification of user rationale. To that end, we used the truth set as the input for the supervised machine learning techniques in order to classify the concepts of user rationale in user reviews.

### 4.2.3. Usage Scenarios

The OpenReq trial partners have large corpora of user comments that provide an important resource for them. Our approach might be employed to a) structure software users' comments; b) filter and synthesize rationale-backed comments for certain stakeholders, such as developers, analysts, or other users; c) improve communication among stakeholders.

In particular, user rationale classifiers might be employed to structure user comments by highlighting polarizations and the broad spectrum of rationale and perspectives in their

discussions. This would enable stakeholders to quickly survey the growing number of user feedback and to help users to easily provide feedback or to get involved into existing discussions. User rationale classifiers can be also used for visualization of pro- and contra-users' stances which highlight contrasting user perceptions on, for example, non-functional requirements (such as pro and contra stances on software usability).

For community managers, user rationale can be used to mark, filter, and synthesize potentially low/highly informative comments. Those, for instance, that have a low justification density can be filtered out. Furthermore, rationale-backed comments of special interest can be selected and explored—e.g., comments mentioning conclusive decisions on switching to a new software product. This allows stakeholders to inspect the best user rationale concepts which can be highlighted or searched for within the comments. User rationale classifiers can help in requirements prioritization and negotiation. For instance, an issue can be highly prioritized when it is frequently mentioned along with a reason for abandoning a software product. Alternatives can give insights on how software features are used together and which compatibility requirements should be considered.

The broad spectrum of their justifications can help to get insights and to better understand users' different perspectives (e.g., on usability). In particular, the whole communication among stakeholders can be improved—e.g., by reusing users' arguments during requirements negotiation. Furthermore, users' justifications can enrich existing requirements documentation and design decisions.

# 5. Summary

In this document, we first presented the idea of requirements intelligence (RI) which gathers data from different sources and from different perspectives to suggest new requirements and suggest revisions for existing ones based on different types of feedback (i.e., implicit and explicit).

The visualization of the RI can be realized using open source tools, such as BIRT. Such tool already ships a lot of diverse dashboard functionalities to present reports of the processed data.

We proposed the use of crawlers and open source libraries to mine explicit and implicit data. The processing will be realized in multiple steps according to the current state of the art in machine learning, data mining and natural language processing applied to requirements engineering.

We proposed a microservice architecture, divided into three layers (i.e., data collection, data storage, and data analytics), to enable highly decoupled software components each focusing on a small set of tasks. These components expose their APIs which can be used by third parties and, thus, can be easily integrated with other components of the OpenReq platform.

Finally, we discussed two papers, related to this deliverable, that had been published as a result of the current undergoing research in the context of OpenReq.

# 6. References


[1] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. 2015. Automated checking of conformance to requirements templates using natural language processing. IEEE transactions on Software Engineering 41, 10 (2015), 944–968.

[2] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. 2008. Duplicate bug reports considered harmful… really?. In 2008 IEEE International Conference on Software Maintenance. 337–345. https://doi.org/10.1109/ICSM.2008.4658082

[3] F. Boltužić and J. Šnajder. 2014. Back up your Stance: Recognizing Arguments in Online Discussions. In Proceedings of the First Workshop on Argumentation Mining. Association for Computational Linguistics, Baltimore, Maryland. http://www.aclweb.org/anthology/W/W14/W14-2107

[4] B. Bruegge and A. A. Dutoit. 1999. Object-Oriented Software Engineering; Conquering Complex and Changing Systems. Prentice Hall.

[5] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrk. 2008. Rationale-Based Software Engineering. Springer.

[6] R. J. G. B. Campello, D. Moulavi, and J. Sander. 2013. Density-Based Clustering Based on Hierarchical Density Estimates. Springer Berlin Heidelberg, Berlin, Heidelberg, 160–172. https://doi.org/10.1007/978-3-642-37456-2_14

[7] N. Carlson and P. Laplante. 2014. The NASA Automated Requirements Measurement Tool: A Reconstruction. Innov. Syst. Softw. Eng. 10, 2 (June 2014), 77–91. https://doi.org/10.1007/s11334-013-0225-8

[8] A. Casamayor, D. Godoy, and M. Campo. 2010. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. Information and Software Technology 52, 4 (2010).

[9] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. 2014. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). ACM, New York, NY, USA, 767–778. https://doi.org/10.1145/2568225.2568263

[10] L. Chung and B. A Nixon. [n. d.]. Dealing with non-functional requirements: three experimental studies of a process-oriented approach. In 17th International Conference on Software Engineering, ICSE 1995. IEEE.

[11] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. 2010. A machine learning approach for tracing regulatory codes to product specific requirements. In Proc. of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM.

[12] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. 2006. The detection and classification of non-functional requirements with application to early aspects. In Requirements Engineering, 14th IEEE International Conference. IEEE.

[13] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. 2011. On-demand Feature Recommendations Derived from Mining Public Product Descriptions. In Proceedings of the 33rd International

Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 181–190. https://doi.org/10.1145/1985793.1985819

[14] A. H Dutoit, R. McCall, I. Mistrik, and B. Paech. 2006. Rationale Management in Software Engineering. Springer, Berlin, Heidelberg.

[15] J. Eckhardt, A. Vogelsang, and D. Méndez Fernández. 2016. Are non-functional requirements really non-functional?: an investigation of non-functional requirements in practice. In Proc. of the 38th International Conference on Software Engineering. ACM.

[16] M. Glinz. 2007. On non-functional requirements. In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International. IEEE.

[17] E. Guzman and W. Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In 2014 IEEE 22nd International Requirements Engineering Conference (RE). 153–162. https://doi.org/10.1109/RE.2014.6912257

[18] N. Jalbert and W. Weimer. 2008. Automated duplicate detection for bug tracking systems. In 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN). 52–61. https://doi.org/10.1109/DSN.2008.4630070

[19] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher. 2013. Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings. IEEE Transactions on Software Engineering 39, 12 (Dec 2013), 1736–1752. https://doi.org/10.1109/TSE.2013.39

[20] M. Harman, Y. Jia, and Y. Zhang. 2012. App Store Mining and Analysis: MSR for App Stores. In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR '12). IEEE Press, Piscataway, NJ, USA, 108–111. http://dl.acm.org/citation.cfm?id=2664446.2664461

[21] A.P.J. Jarczyk, P. Loffler, and F.M. Shipman. [n. d.]. Design rationale for software engineering: a survey. In System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on. https://doi.org/10.1109/HICSS.1992.183309

[22] T. Johann, C. Stanik, A. M. Alizadeh, and W. Maalej. 2017. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In 2017 IEEE 25th International Requirements Engineering Conference (RE) (2017-08-01). to appear.

[23] L. Kof. 2004. Natural Language Processing for Requirements Engineering: Applicability to Large Requirements Documents. (09 2004).

[24] G. Kotonya and I. Sommerville. 1998. Requirements engineering: processes and techniques. Wiley Publishing.

[25] J. Lee. 1997. Design Rationale Systems: Understanding the Issues. IEEE Expert: Intelligent Systems and Their Applications 12, 3 (May 1997). https://doi.org/10.1109/64.592267

[26] Y. Liang, Y. Liu, C. K. Kwong, and W. Bun Lee. 2012. Learning the "Whys": Discovering design rationale using text mining—An algorithm perspective.

Computer-Aided Design 44, 10 (2012), 916–930.

[27] M. Lippi and P. Torroni. 2016. Argumentation mining: State of the art and emerging trends. ACM Transactions on Internet Technology (2016).

[28] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. 2016. On the Automatic Classification of App Reviews. Requir. Eng. 21, 3 (Sept. 2016), 311–331. https://doi.org/10.1007/s00766-016-0251-9

[29] L. Mich, M. Franch, and P. Novi Inverardi. 2004. Market Research for Requirements Analysis Using Linguistic Tools. 9 (01 2004), 40–56.

[30] D. Pagano and W. Maalej. 2013. User feedback in the appstore: An empirical study. In 2013 21st IEEE International Requirements Engineering Conference (RE). 125–134. https://doi.org/10.1109/RE.2013.6636712

[31] R. M. Palau and M.-F. Moens. 2009. Argumentation Mining: The Detection, Classification and Structure of Arguments in Text. In Proceedings of the 12th International Conference on Artificial Intelligence and Law (ICAIL '09). ACM, New York, NY, USA, 98–107. https://doi.org/10.1145/1568234.1568246

[32] A. Peldszus and M. Stede. 2013. From Argument Diagrams to Argumentation Mining in Texts: A Survey. International Journal of Cognitive Informatics and Natural Intelligence (IJCINI) 7, 1 (2013), 1–31. https://doi.org/10.4018/jcini.2013010101

[33] B. Rogers, J. Gung, Yechen Qiao, and J.E. Burge. 2012. Exploring techniques for rationale extraction from existing documents. In Software Engineering (ICSE), 2012 34th International Conference on. https://doi.org/10.1109/ICSE.2012.6227091

[34] J. Slankas and L. Williams. 2013. Automated extraction of non-functional requirements in available documentation. In 1st International Workshop on Natural Language Analysis in Software Engineering. IEEE.

[35] I. Sommerville and P. Sawyer. 1997. Requirements Engineering: A Good Practice Guide (1st ed.). John Wiley & Sons, Inc.

[36] R. B. Svensson, T. Gorschek, and B. Regnell. 2009. Quality requirements in practice: An interview study in requirements engineering for embedded systems. In International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer.

[37] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. 2016. Release Planning of Mobile Apps Based on User Reviews. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). ACM, New York, NY, USA, 14–24. https://doi.org/10.1145/2884781.2884818

[38] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. 2001. Constrained K-means Clustering with Background Knowledge. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 577–584.

[39] T. Johann, G. Ruhe, W. Maalej, M. Nayebi. 2016. Toward Data-Driven Requirements Engineering. IEEE Software: Special Issue on the Future of Software Engineering 33, 1 (2016), 48–54.

[40] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In 2008 ACM/IEEE 30th International Conference on Software Engineering. 461–470. https://doi.org/10.1145/1368088.1368151

[41] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. 1997. Automated Analysis of Requirement Specifications. In Proceedings of the (19th) International Conference on Software Engineering. 161–171. https://doi.org/10.1145/253228.253258

[42] W. Maalej . Kurtanović. 2017. Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning. In 2017 IEEE 25th International Requirements Engineering Conference (RE) (2017-08-01). to appear.

[43] W. Maalej Z. Kurtanović. 2017. Mining User Rationale from Software Reviews, In 2017 IEEE 25th International Requirements Engineering Conference (RE) (2017-08-01). 2017 IEEE 25th International Requirements Engineering Conference (RE), to appear.

[44] M. Bates. 1995. Models of natural language understanding. Proceedings of the National Academy of Sciences of the United States of America, 92(22), 9977–9982.

[45] L. M Davis. 1993. Software requirements: objects, functions, and states. Prentice-Hall, Inc.

[46] S. Bird, E. Klein, and E. Loper. 2009. Natural Language Processing with Python (1st ed.). O'Reilly Media, Inc.

[47] D. Martens and T. Johann. 2017. On the Emotion of Users in App Reviews. InProceedings of the 2Nd International Workshop on Emotion Awareness in Software Engineering(SEmotion '17). IEEE Press, Piscataway, NJ, USA, 8–14.

[48] R. Basili, C. Bosco, R. Delmonte, A.Moschitti & M. Simi (2015). Harmonization and development of resources and tools for Italian natural language processing within the PARLI Project. Cham: Springer.

[49] I. Rehbein, J. Ruppenhofer, C. Sporleder, M. Pinkal. Adding nominal spice to SALSA - frame-semantic annotation of German nouns and verbs . In Proceedings of KONVENS 2012, Vienna, Austria, Sept. 19-21, 2012.

[50] M. Verma, M. Srivastava, N. Chack, A. K. Diswar, and N. Gupta. 2012. A comparative study of various clustering algorithms in data mining. International Journal of Engineering Research and Applications (IJERA) 2, 3 (2012), 1379–1384.

[51] Y. S. Van Der Sype and W. Maalej. 2014. On lawful disclosure of personal user data: What should app developers do?. In 2014 IEEE 7th International Workshop on Requirements Engineering and Law (RELAW) . 25–34. https://doi.org/10.1109/RELAW.2014.6893479

[52] C. Stanik, N. Mannov, D. Weiser, W. Maalej, "D6.2-First Prototype of the user observation, application monitoring tools and context actuators," Mar. 2014.

[53] Y. Yang, P. Nobel, R. Wu, H. Zhang, D. Zhang (2012) ReBucket: A method for clustering duplicate crash reports based on call stack similarity. 34th International Conference on Software Engineering (ICSE), 1084-1093

[54] Anvik, J., Sun, J., Wang, X., Xie, T., & Zhang, L. (2008). An approach to detecting duplicate bug reports using natural language and execution information. 2008 ACM/IEEE 30th International Conference on Software Engineering, 461-470.

[55] S. Rastkar, G.C. Murphy, G. Murray (2010). Summarizing software artifacts: a case study of bug reports. 2010 ACM/IEEE 32nd International Conference on Software Engineering, 1, 505-514.

[56] G. Carenini and G. Murray (2008). Summarizing Spoken and Written Conversations. *EMNLP*.

[57] J. Yiang, S. Khoo, D. Lo, C. Sun and X Wang. (2010). A discriminative model approach for accurate duplicate bug report retrieval. 2010 ACM/IEEE 32nd International Conference on Software Engineering, 1, 45-54.

[58] J.R. Cooper Jr., S. Lee, R.A. Gandhi and O. Gotel (2009). Requirements Engineering Visualization: A Survey on the State-of-the-Art. 2009 Fourth International Workshop on Requirements Engineering Visualization (rev'09).

[59] C. Jesdabodi and W. Maalej. 2015. Understanding Usage States on Mobile Devices. In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15) . ACM, New York, NY, USA, 1221–1225. https://doi.org/10.1145/2750858.2805837

[60] Y. Zhan, S. Wang, Z. Zhao, C. Chen, and J. Ma. 2009. A mobile device oriented framework for context information management. In 2009 IEEEYouth Conference on Information, Computing and Telecommunication. 150–153. https://doi.org/10.1109/YCICT.2009.53824041

[61] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas (2010). Sentiment strength detection in short informal text. Journal of the American Society for Information Science and Technology, 61(12), 2544–2558.

[62] Z. S. H. Abad, M. Noaeen, and G. Ruhe. 2016. Requirements Engineering Visualization: A Systematic Literature Review. In 2016 IEEE 24th International Requirements Engineering Conference (RE). 6–15. https://doi.org/10.1109/RE.2016.61

[63] A. Strauss and J. Corbin. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. SAGE, 1998.