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

D3.1 OpenReq Approach for Stakeholders' Recommendations

Lead contractor: UPC

Author(s): HITEC, TUGraz, UPC

Submission date: December 2017

Dissemination level: PU

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

Abstract: This deliverable presents a description of the state-of-the-art in recommender systems, both from a scientific and practical point of view. Moving from this state-of-the-art, the deliverable also states the requirements for the stakeholders' recommendations in OpenReq. Specifically, it defines the different stakeholders' recommendations tasks that have been identified (stating for each one of them the requirements artifacts that are involved), the technical approach and algorithms used to implement the recommendation tasks, and the general architecture of the stakeholders' recommendations in OpenReq. Some of the initial results are summarized that will be refined later over the course of the project. The results include the surveys of recommender systems and requirements quality improvement, and the architecture and data model to be used.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

One of the stated objectives of the OpenReq project is to design an approach for assisting individual stakeholders in different requirements-related tasks such as defining, reusing, screening, understanding, evaluating, and quality assurance. To achieve that, the first actions covered in this document are:

- To carry out a state-of-the-art in recommender systems, both from a scientific and practical point of view.
- To identify the stakeholders' recommendations tasks and their related requirement artifacts.
- To define the technical approach and algorithms used to implement the recommendation tasks.
- To define the general architecture of the stakeholders' recommendations in OpenReq.

To this end, in this deliverable we show the current state of these four actions.

1.1 Motivation

During the formulation of the proposal, we stated the need for having recommendations that help stakeholders, as individuals, during the requirements engineering process. These recommendations are related to the screening and recommendation of relevant requirements, to the improvement of requirements quality, to the prediction of requirements properties, and to the identification of relevant stakeholders. In this deliverable, we further analyse and explain the design of the stakeholders' recommendations approach required to cover the expectations of the OpenReq project and its intended platform.

1.2 Glossary of terms

The following table presents the most used terms in the document.

Table 1. List of terms used in the document

Term	*Description*
CARS	Context-Aware Recommender System (as a type of recommender system)
CF	Collaborative Filtering (as a type of recommender system)
CBF	Content-Based Filtering (as a type of recommender system)
HRS	Hybrid Recommender System (as a type of recommender system)
KB	Knowledge-Based (as a type of recommender system)
ML	Machine Learning
RE	Requirements Engineering
RS	Recommender System
SR	Social Recommender (as a type of recommender system)

1.3 Intended audience

The content of this document is of special interest for the OpenReq consortium, because it defines stakeholders' recommender engine that is one of the four aspects covered on the OpenReq project. We also plan to disseminate parts of this work to the research community, especially the systematic mappings done for the literature reviews about recommender systems and requirement quality improvements.

1.4 Relation to other deliverables

This deliverable is greatly related to other deliverables of the project, mainly with:

- D2.1, which presents the OpenReq approach for analytics and requirements intelligence;
- D4.1, which presents the OpenReq approach for group decision support; and
- D5.1, which presents the OpenReq approach for requirements knowledge and dependency management.

1.5 Scope

This deliverable will evolve as the project and work package activities progress and more consolidated versions of the design are in place, especially for those tasks that have not yet started (T3.4, T3.5 and T3.6). Here, we explain in detail the stakeholder requirements recommendations strategy for OpenReq and the current status of its design.

1.6 Document structure

Section 1 introduces the deliverable following the OpenReq deliverable template. Section 2 presents the state-of-the-art in recommender systems and requirements quality improvement. Section 3 describes the different stakeholders' recommendations tasks that have been identified, the technical approach and algorithms used to implement the recommendation tasks, and the general architecture of the stakeholders' recommendations in OpenReq. Section 4 presents a conclusion of the work reported in this deliverable.

2 STATE-OF-THE-ART

This section summarizes the state-of-the-art related to OpenReq personal recommendations for stakeholders. We summarize the topic of recommender systems (subsection 2.1) and the topic of requirement quality improvement (subsection 2.2) since they are relevant for the work done in OpenReq personal recommendations for stakeholders and could inspire the whole approach. Finally, some conclusions of the state-of-the-art and practice are presented (subsection 2.3).

2.1 Recommender systems

Recommender Systems (RS) support users in finding items of interest. The major goal of this subsection is to present the basic properties of the three major recommendation approaches: collaborative filtering (subsection 2.1.1), content-based filtering (subsection 2.1.2) and knowledge-based (subsection 2.1.3). Thereafter, we describe the properties of hybrid recommendation approaches which combine some of the basic variants listed above (subsection 2.1.4), and a summary of other techniques that are being used in recommendation systems (subsection 2.1.5). In addition to introducing the recommendation approaches, subsection 2.1.6 also includes a summary of how recommendation systems deal with context-awareness. Figure 1 summarizes this different approaches to recommender systems that will be explored in this subsection.

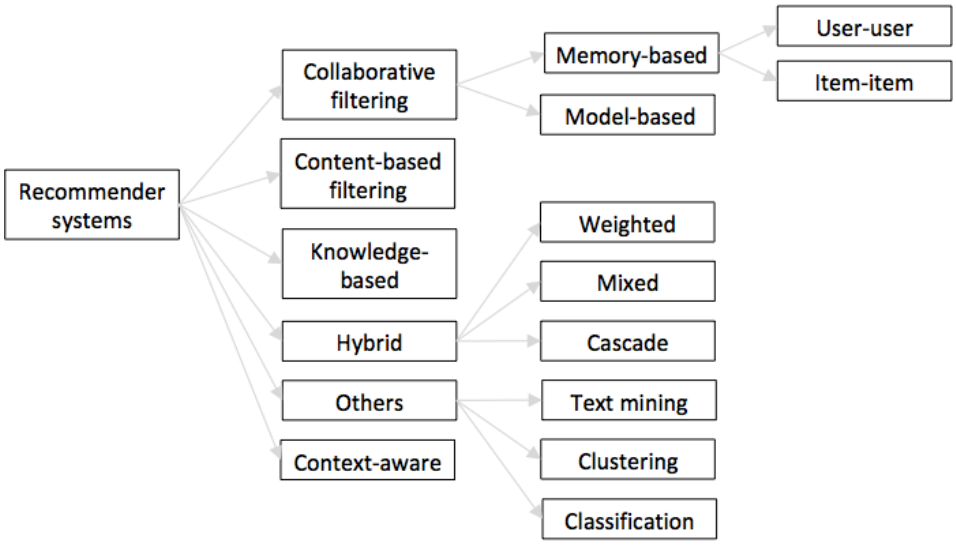

Figure 1. Recommender systems classification

Apart from the main properties of these recommendation techniques, these subsections also include the most relevant approaches for developing the personal recommendations for stakeholders in OpenReq. These approaches have been found doing a systematic mapping following a snowballing process which used as seed the work of Jannach (2010). As stated before, we only plan to cite here the most relevant approaches for the development of OpenReq. The full snowballing results (which include more than 90 papers) are aimed to be included in a publication. Therefore, we will not mention in this section some works identified about the application of RS in practice (e.g., Netflix in (Meuth 2008) and (Bell 2009), and Amazon in (Linden 2003)) since:

1. The recommendations of these RS are quite different from the ones to be done in OpenReq; and

2. As these RS are usually trade-market secrets, the few publications found probably only explain just a small part of the RS used in these commercial systems.

2.1.1 Collaborative filtering (CF) recommendation approaches

Collaborative filtering (CF) is a recommendation technique that bases its predictions and recommendations on the past ratings or behavior of other system users (Ekstrand 2011). The assumptions behind this technique are (Bernardes 2015): first, if users agree about the quality or relevance of some items, then they will likely agree about other items; and second, users' preferences remain stable and consistent over time. The majority of CF algorithms operate by first generating predictions of the user's preference and then produce their recommendations by ranking candidate items by predicted preferences.

Two main groups of CF algorithms exist (Felfernig 2014): *memory-based algorithms* (also known as kNN), and *model-based algorithms* (also known as latent factor models). Memory-based algorithms operate over the entire user data to make predictions (e.g., (Castro-Herrera 2009, 2010), (McCarey 2005)). In contrast, model-based algorithms use the user data to build a model which is then used for recommendations (e.g., (Breese 1998), (Hofmann 2004)). Memory-based algorithms are simpler, seem to work reasonably well in practice and new data can be added easily (Felfernig 2014). For this reason, the rest of this subsection focuses mainly on memory-based algorithms.

Memory-based algorithms have two major branches: *user-user algorithms* and *item-item algorithms*.

User-user algorithms (e.g., (Lim 2012), (Zhang 2013)) use a direct algorithmic interpretation of the first assumption of CF: find other users whose past rating behavior is similar to that of the current user, and use their ratings on other items to predict what the current user will like. This type of algorithms need:

1. A rating matrix *R*, stating the rating of users over different items.
2. A similarity function computing the similarity between two users (using their ratings), $s: UxU \rightarrow R$. Some typical similarity functions are Pearson, Constrained Pearson, Spearman and Cosine (Ekstrand 2011).
3. A method for using similarities and ratings to generate predictions. In a nutshell, this method uses the similarity function *s* to compute a neighbourhood $N \subseteq U$ of neighbours of a specific user and later it combines the ratings of users in *N* to generate predictions for the preference for an item *i* of the current user. This is typically done by computing the weighted average of the neighbouring users' ratings of item *i*.

Item-item algorithms (e.g., (Verstrepen 2015)) are similar to user-user algorithms, but rather than using similarities between users' rating behavior to predict preferences, they use similarities between the rating patterns of items. The assumption is that if two items tend to have the same users liking and disliking them, then they are similar and users are expected to have similar preferences for similar items. Similarly to user-user algorithms, item-item algorithms need:

1. A rating matrix.
2. A similarity function.
3. A method for using similarities and ratings to generate predictions.

However, in contrast to user-user algorithms, in item-item algorithms the similarity and prediction method are based on items instead of users.

In RS where the number of users exceeds the number of available items, item-based approaches are preferred since they provide more accurate recommendations, while being more computationally efficient and requiring less frequent updates (Ricci 2015).

Several approaches have been proposed to improve memory-based algorithms:

- Alan (2012) uses *weighting schemes* to improve the similarity measures used in this type of RS and avoiding the bias of items that are rated by a lot of users. To achieve it, similarities are computed taking the popularity of the set of two users' co-rated items into consideration, e.g., an item rated by a large number of users should have less impact on the similarity measure, and analogously an item rated by few users should have a larger impact on the similarity score of two users.
- Tho-Sutter (2008) proposes a generic algorithm that uses *tags*, which are "local" descriptions of items given by the users.

Another branch of research to improve memory-based algorithms is the incorporation of social media information, generally by adding this information as another input of the CF algorithm and using this social media information during the similarity measure and prediction process. Some example are:

- Zhang (2013) proposes, in the e-commerce domain, a new prediction algorithm based on using social media information and the brands the users liked in these social media channels.
- Lim (2012) presents the *StakeRare* method, which uses CF and social networks to identify and prioritize requirements in large software projects. StakeRare identifies stakeholders and asks them to recommend other stakeholders and stakeholder roles, builds a social network with stakeholders as nodes and their recommendations as links, and prioritizes stakeholders using a variety of social network measurements to determine their project influence. The method then asks the stakeholders to rate an initial list of requirements, recommends other relevant requirements to them using CF, and prioritizes their requirements using their ratings weighted by their project influence.

The use of social media has given place to a new type of recommender, called *Social Recommenders* (SR), which are discussed in subsection 2.1.4.

One problem of memory-based algorithms is the well-known *cold start problem*, i.e., providing recommendations when there is not yet data available on which to base the predictions, because either the user or the item are new in the system (Ekstrand 2011). Another problem of CF algorithms is the *sparsity* of the rating matrix because most of the users rate only few items and, hence, the user-item rating matrix is typically very sparse (Mellville 2002). This implies that the probability of finding a set of users with significantly similar ratings is usually low, especially in systems that have a very high item-to-user ratio or when the system is in the initial stage of use.

There exist several works that try to mitigate the cold-start-problem:

- Adamopoulos (2013) proposes the use of weighting schemes in memory-based algorithms, where the estimation of an unknown user rating for an item is based not on

the weighted average of the *k* nearest neighbours but on the weighted percentile of the ratings of these *k* neighbours.

- Fernández (2016) evaluates different approaches, based on exploiting user personality, to solve the problem.
- Rashid (2002) also evaluates different approaches to solve the problem, but in this case the approaches are based on the use of information theory, statistics aggregation and personalization.

In (Pennock 2000), both memory-based and model-based approaches are combined, resulting in a new CF algorithm that they call *personality diagnosis* (PD). The idea behind PD is that, given the user's preferences for some items, it computes the probability that this user belongs to the same "personality type" as other users and, in turn, the probability that this user will like new items. PD retains some of the advantages of traditional memory-based algorithms like the fact that all data is brought to bear on each prediction and new data can be added easily and incrementally, but it also has a meaningful probabilistic interpretation, which may be leveraged to justify, explain, and augment results.

Some CF approaches have been proposed specifically for the Software Engineering field, and in particular for the RE field:

- Lim (2012), introduced some paragraphs above, is one of these approaches. It focuses on identifying and prioritizing requirements.
- McCarey (2005) presents the *RASCAL* recommender system. RASCAL aims to predict the next method that a developer could use, by analysing classes similar to the one currently being developed. RASCAL's "users" are classes and the items to be recommended are methods to be called. The similarity between the current class and other classes is essentially based on the methods they call.
- Gaeul (2009) proposes a more advanced type of CF algorithm based on Markov chains to improve the "tossing" (reassignment) of bugs to other developers, for example because the bug has been assigned incorrectly to a developer or another developer with additional expertise is needed. The approach incorporates a graph model based on Markov chains, which captures bug tossing history. This model reveals developer networks which can be used to discover team structures and to find suitable experts for a new task and helps to better assign developers to bug reports.
- Castro-Herrera (2009, 2010), focused on the RE field, aims to improve the requirements elicitation process of large and distributed software projects. In order to achieve this, it presents different CF algorithms to inform individual stakeholders of relevant forums, where particular types of requirements are discussed, that might be of interest for them.

2.1.2 Content-based filtering (CBF) recommendation approaches

In the RS based on *Content-Based Filtering* (CBF) (Felfernig 2014), a user gets recommendations for items similar to the ones she preferred in the past, in contrast to CF techniques where the user will be recommended items that people with similar tastes and preferences liked in the past. CBF is based on the assumption of monotonic personal interests. CBF techniques need (Adomavicius 2005) (Felfernig 2014):

1. A set of users, usually with the addition of user profiles that contain information about users' tastes, preferences, and needs.
2. A set of categories (or keywords) that have been assigned to (or extracted from) the available items (item descriptions).
3. A utility function that calculates a set of items that are most similar to items already known to the current user. Therefore, the results of the utility function $u(a, i)$ of item i for user a is estimated based on the utilities $u(a, i_j)$ assigned by user a to items $i_j \subset I$ that are "similar" to item i (excluding i itself). For instance, k-nearest neighbors could be used as a utility function.
4. A similarity function, which is based on keywords extracted from the item descriptions or categories in the case that items have been annotated with the relevant meta-information. The major difference from the similarity metrics of CF techniques is that in this case similarity is measured using keywords (in contrast to ratings). Therefore, several similarity metrics used in natural language processing can be used, such as Cosine, Dice, and Jaccard (Natt och Dag 2001).

As a result, only the items that have a high degree of similarity to whatever the user's preferences are would be recommended to the user.

A basic CBF approach is presented in (Musto 2010), which is based on *Vector Space Models* (VSM), an established technique in the area on information retrieval. In VSM, each document is represented by a vector in a n-dimensional space, where each dimension corresponds to a term from the overall vocabulary of a given document collection. VSM has two advantages:

1. Its very clean and solid formalism allows to represent objects in a vector space and to perform calculations on them.
2. Although being a simple algorithm, it is a very effective model (Basile 2010).

Additionally, Musto (2010) introduces two approaches for improvement:

1. One approach is based on random indexing, which implements a scalable and effective VSM-based approach.
2. Another approach introducing a negation operation in order to overcome the classical VSM problem that arises from the impossibility to manage the evidences about negative preferences.

More advanced semantics techniques can be applied in CBF algorithms, as explained in (de Gemmis 2015). Some of the proposals presented there aim to:

- Incorporate ontological knowledge, ranging from simple linguistic ontologies, to more complex domain-specific ones;
- Leverage unstructured or semi-structured encyclopaedic knowledge sources, such as Wikipedia; and

- Exploit the wealth of the so-called Linked Open Data cloud[1].

An approach for recommending books based on CBF algorithms is presented in (Mooney 2004), which utilizes information extraction and a machine-learning algorithm for text categorization. The learning of user profiles can be cast as a binary text categorization task: each document (i.e., book) has to be classified as interesting or not with respect to the user preferences. More specifically, the proposed system uses a database of book information extracted from an e-commerce web distributor. Users provide 1-10 ratings for a selected set of books, and then the system learns a profile of the user using a bayesian learning algorithm and produces a ranked list of the most recommended additional titles from the system's catalogue.

In the RE field, CBF recommendations approaches have also been used:

- Dumitru (2011) presents a CBF recommendation approach to requirements reuse. The basic idea is to analyse requirements which are accessible in software project repositories and to apply clustering techniques for the intelligent grouping of such requirements. The identified requirement groups can be analysed in future software projects for the purpose of reuse and also for the purpose of completeness checking (i.e., are all relevant requirements contained in the current requirements model). The proposed recommendation approach uses a vector of keywords (derived from the description of the new software project) which is matched with the keywords extracted from requirements artifacts from the repository of already completed software projects.
- Castro-Herrera (2009, 2010) shows how to exploit clustering techniques for grouping user requirements and in the following to recommend stakeholders to clusters on the basis of CBF algorithms.

2.1.3 Knowledge-based (KB) recommendation approaches

Knowledge-based (KB) recommender systems (Felfernig 2014), compared to CF and CBF that primarily rely on item ratings and textual item descriptions, exploit deep knowledge (semantic knowledge) about the item in order to determine recommendations. KB techniques need (Felfernig 2009, Felfernig 2013, Felfernig 2014):

1. Explicit knowledge about the given set of user requirements.
2. Deep knowledge about the underlying items and their properties.
3. Recommendation knowledge represented in the form of explicit constraints that relate requirements to the corresponding item properties, or similarity metrics to select items that are most similar to the user requirements. In the case of similarity metrics, attribute-level similarity measures are predominantly applied.
4. A ranking algorithm to order the possible recommendations. One widespread approach to rank items is to define a utility scheme which serves as a basis for the application of Multi-Attribute Utility Theory (MAUT), where items can be evaluated and ranked with respect to a defined set of interest dimensions.

[1] http://lod-cloud.net/

Therefore, interacting with a KB recommender system typically means to (Felfernig 2009):

1. Answer a set of questions (requirements elicitation phase).
2. Repairing inconsistent requirements (if no recommendation could be found).
3. Evaluating recommendations.

KB algorithms do not suffer from the ramp-up problem since their recommendations do not depend on having a base of user ratings. In addition, they do not have to gather information about a particular user because its judgements are independent of individual tastes (Burke 2000).

Felfernig (2006) presents a KB recommender system which is domain-independent. It provides an explicit representation of product, marketing and sales knowledge, which allows:

1. To calculate solutions which adhere to legal regulations, which are in line with a company's marketing and sales strategy, and which suit to the requirements of the customer.
2. To explain solutions to a customer.
3. To support customers in situations in which no solution can be found.

The recommender dialogues in (Felfernig 2006) are based on a finite state model that describes possible interaction sequences of a recommender system on a graphical level. Using such representations, the formulation of questions, answers, and explanations can be automatically adapted to the domain knowledge level and preferences of a user.

Dialogues in KB recommender systems are the focus of (Burke 1996). This work presents the use of assisted browsing to allow accessing to information along a multitude of dimensions and from a multitude of sources without the user needing to be aware of this complexity. The approach has two basic parts: 1) an initial query stage through which users state their starting point within the information space, and 2) an assisted browsing phase in which users traverse the information space. Explanations, similarity-based retrieval and tweaking are part of a dialogue between the user and the system in which the user comes to a better understanding of the domain by using examples (through learning about trade-offs and looking at many examples) and the system helps the user finding specific items of interest by gradually refining the goal.

In the area of RE, several approaches propose the use of KB algorithms:

- Romero (2004) proposes a security requirements RS to recommend an appropriate approach to security for a specific project. Before the recommendation process starts, the RS needs information about the security approaches it will recommend (i.e., a number of how much an approach fulfils specific security characteristics). During the recommendation process, the system takes user input about the most desirable characteristics for the security of the project (in the form of ratings) and recommends the most appropriate security approach comparing the user' input with the information of the approaches.
- Kumar (2010) presents a RS for knowledge assisted agile requirements evolution. The aim is to incorporate domain knowledge to achieve agility to the requirements definition stage by providing requirement analysts with online domain specific recommendations based on underlying ontologies. The framework presents a 'domain

knowledge seed' to requirement analysts. This seed provides a view of core features in a given domain and associated knowledge elements such as business processes, rules, policies, partial data models, use cases and test cases. These in turn are mapped with agile requirements elements such as user stories, features, tasks, product backlog, sprints and prototype plans. Requirement analysts can evolve the seed to suit their specific project needs. As they modify and evolve the seed specification, they receive domain-specific online recommendations to improve the correctness, consistency and completeness of their requirements specification documents and executable models.

- Felfernig (2013) develops an approach to determine personalized diagnoses for inconsistent requirements in KB recommendation scenarios. Specifically, the repairs are proposed when no item can be recommended due to having too many constraints. The repair proposals are generated on the basis of five different search strategies:
    a) cardinality-based (i.e., detecting minimal sets of conflicting requirements),
    b) similarity-based (i.e., identifying minimal diagnoses which lead to solutions that resemble the original set of requirements as much as possible),
    c) utility-based (i.e., searching for minimal repairs that are predominantly composed of requirements which are of low importance for the user),
    d) probability-based (i.e., detecting minimal diagnoses with a high probability of being selected by the user), and
    e) ensemble-based (i.e., exploiting a set of hypotheses (an ensemble) for making the predictions, in contrast to the previous strategies, where diagnosis predictions are based on a single hypothesis).

2.1.4 Hybrid recommendation system (HRS) approaches

Hybrid recommender systems (HRS) are based on the combination of some of the techniques introduced in the previous subsections (Burke 2002). The motivation of combining different recommender techniques is twofold:

1. To achieve better recommendations (Felfernig 2014); and
2. To overcome one disadvantage of one of the techniques being used (Ricci 2015).That is, a HRS combining techniques A and B tries to use the advantages of A to fix the disadvantages of B. For instance, CF techniques suffer from new-item problems (i.e., they cannot recommend items that have no ratings), but this is not the case in CBF techniques since the prediction for new items is based on their description (features), which are typically easily available. Therefore, combining CF and CBF is a good idea when trying to overcome the new-item problem.

Given two (or more) basic RS techniques, several ways have been proposed for combining them to create a new HRS (Burke 2002):

1. *Weighted* HRS, which is based on the idea of deriving recommendations by combining the results computed by individual recommenders. For instance, implementing CF and CBF techniques separately, and combining (e.g., summing up) their predictions later on.
2. *Mixed* HRS, which main principle is the idea that predictions of individual recommenders are shown in one integrated result. One example could be implementing

CF and CBF techniques separately, and integrating the results into one score using some principle, such as the zipper principle (i.e., best CF prediction receives highest score, best CBF prediction receives the second highest score, second best CF prediction receives third highest score, and so forth).

3. *Cascade* HRS, which core idea is that recommenders in a pipe of recommenders exploit the recommendation of the upstream recommender as a basis for deriving their own recommendation. For instance, implementing a CF technique and then using these predictions as one further input for a CBF technique.

Earlier HRS often internally combine two classical techniques, usually CF and CBF (e.g., Melville 2002). Given that in these types of HRS specific properties of the individual techniques are exploited, they lack extendibility and easy integration of other types of techniques (Dooms 2013). In (Melville 2002), CF and CBF techniques are combined to overcome the shortcomings of CF. They provide a framework for combining both approaches, which uses a CBF predictor to enhance existing user data (i.e., to convert a sparse user ratings matrix into a full ratings matrix), and then provides personalized recommendations through CBF. In contrast, HRS using individual recommendation techniques as black boxes (e.g., (Dooms 2013)) allow to easily extend their models with other techniques of any type (Dooms 2013).

A branch that has been explored inside HRS is to combine more classical recommendation techniques with the so-called *Social Recommenders* (SR) (i.e., any RS targeting social media artefacts, such as blogs, social tagging, video sharing, etc.) (Guy 2015):

- Bernardes (20015) proposes a framework for constructing HRS based on social network analysis. This framework allows to describe association rules for both traditional CF recommenders and SR.
- Basu (1998) presents an inductive learning approach to recommendation that is able to use both ratings information (coming from SR) and other forms of information (coming from CBF techniques) about each item in predicting user preferences. In particular, users' ratings are used in the inductive learning process to create hybrid features. Hybrid features are social features that are influenced by content. For instance, in the domain of movies recommendation, a hybrid feature would be *users who liked dramas*, reflecting both information of users' likes (from SR) and a specific genre (from CBF).

Finally, several advanced frameworks have been proposed to create HRS, such as (Dooms 2013) and (Hussein 2014), with the aim of easing the process of building and tuning a HRS, which usually is tedious and time-consuming:

- Dooms (2013) focuses on dynamically building personalized HRS on an individual user basis by a means of a dynamic online learning strategy that combines the most appropriate recommendation techniques for a user based on real-time relevance feedback. The framework integrates over 20 techniques from the recommendation framework MyMediaLite[2].

[2] http://www.mymedialite.net/

- Hussein (2014) introduces a software framework for building complex hybrid, context-aware recommender systems. The framework provides a range of recommendation techniques and strategies for producing group recommendations, templates for combining different methods into HRS, and a means for integrating existing user or product data from external sources such as social networks. The pre-implemented recommenders include:
    a) a variety of techniques provided by Apache Mahout[3] (an item-based and a user-based CF, an item-average recommender, a recommendation algorithm that randomly selects a set of items, and a slope-one recommender),
    b) two variations of spreading activation-based recommenders as examples of CBF algorithms, and
    c) a rule-based recommender.

2.1.5 Other types of recommender approaches

Apart from more classical types of recommenders introduced in the previous sections, in the recent years other techniques from the area of data mining have been incorporated in RS. Here we discuss three of them: *text mining*, *clustering* and *classification*. We focus only on those approaches related to Software Engineering and RE fields discovered in the literature snowballing carried out, since they are of special relevance to OpenReq.

2.1.5.1 Text mining

The purpose of *text mining* is to process unstructured textual information and extract meaningful numerical indices from the text, in order to make the information contained in the text accessible to the different data mining algorithms (statistical and machine learning) (Aggarwal 2012).

Inside text mining, *similarity detection* (i.e., detection of similar texts by using either their syntactic or semantic properties) is an established field. In (Weiß 2007), similarity is used to automatically predict the fixing effort, i.e., the person-hours spent on fixing an issue, such as a software bug. Given a new issue report, the Lucene[4] framework is used to query the database of resolved issues for textually similar reports (using the nearest neighbour approach) and use their average time as a prediction.

Assignments of developers to bug reports has also been tackled from a similarity perspective:

- Olga (2009) presents a framework for automated assignment of bug-fixing tasks which infers knowledge about a developer's expertise by analysing the history of bugs previously resolved by the developer. Then, it applies a *vector space model* (VSM) to recommend experts for fixing bugs, matching the new bug VSM representation with the most similar developer VSM representation. In addition to similarity, other heuristics are taken into account, as current workload and preferences of the developer.

[3] http://mahout.apache.org/

[4] https://lucene.apache.org/core/

- Nagwani (2012) proposes an algorithm to discover experts for fixing new software bugs which is based on the analysis of their textual information (e.g., summary and description attributes). Frequent terms are generated from this textual information and then term similarity is used to identify appropriate experts (developers) for the newly reported software bug.

Text mining is used in combination with machine learning techniques in (Menzies 2008) to assist test engineers in assigning severity levels to defect reports. The proposed algorithm is based on the automated extraction and analysis of textual descriptions from issue reports: text mining techniques are used to extract the relevant features of each report, while machine learning techniques are used to assign these features with proper severity levels (taking into account the severity levels already assigned to other issues to construct rules about when an specific defect level should be assigned).

2.1.5.2 Clustering

Another approach used in RS is *clustering*. Clustering refers to the grouping of a particular set of objects based on their characteristics, aggregating them according to their similarities (Tan 2005a). Different clustering algorithms exists, such as *k*-means, DBSCAN, and HDBSCAN (Verma 2012). *k*-means can easily handle large data but the number of clusters (i.e., the number of topics extracted from the requirements) must be defined a priori. A challenge of defining this number of clusters a priori is to understand the domain and the input. While using HDBSCAN, there is no need to specify the number of clusters explicitly, but it does not scale well and experimentation with data coming from different sources is needed.

Clustering has been used in RSs as a part of CBF techniques (Dumitru 2011) (introduced in the subsection *2.2.2 CBF recommendations approaches*) or by its own (Cleland 2009). An approach to support effective feature management has been introduced by (Cleland 2009) where clusters of similar requirements are exploited for the identification of redundancies and the prioritization of feature requests. In (Chien 2016), clustering is used for topic modelling. It uses structural learning and inference of latent themes and topics for sentences and words from a collection of documents, respectively. The relation between themes and topics under different data groupings is explored through an unsupervised procedure without limiting the number of clusters.

2.1.5.3 Classification

Classification is concerned with identifying a class/category of an unseen observation (Tan 2005b). Classifiers might solve a *binary classification problem* (i.e., classifying the data into only two classes) or a *multi-class problem*. A range of classification algorithms exist, such as support-vector machines, decision trees, and *k*-nearest neighbour.

Antoniol (2008) uses classifiers to decide whether an item in a bug repository is a bug or not. The approach uses a supervised machine learning approach that alternates decision trees, naive bayes classifiers, and logistic regression.

Classifiers have also been used to give value to particular properties of issues in bug repositories, especially the *severity* property of bugs (i.e., the impact the bug has on the successful execution of the software system):

- Fitzgerald (2011) aims at identifying those bugs that may imply an imminent failure in the system. For doing so, the proposed framework automatically constructs failure prediction models using machine learning classification algorithms and allows to

compare the performance of the different techniques included in the framework (i.e., naive bayes, decision tables, linear regression and M5P-Tree).

- Lamkanfi (2010) investigates whether it is possible to accurately predict the severity of a reported bug by analysing its textual description and later on classifying them using a naive bayes classifier, which is based on the probabilistic occurrence of terms (i.e., the features). The assumption behind the approach is that the reporter of a bug uses potentially significant terms in the descriptions which distinguish severe from non-severe bugs. In a follow-up study, Lamkanfi (2011) compares four well-known classification algorithms (namely, naive bayes, naive bayes multinomial, *k*-nearest neighbour and support-vector machines) with respect to accuracy and training set size. For the cases under investigation in the study, naive bayes multinomial performs superior compared to the other proposed algorithms.

Another area where classifiers have been used is the assignment of developers or maintenance teams to new issues in tracker systems:

- Di Luca (2002) proposes to automatically classify incoming tickets, routing them to specialized maintenance teams. The idea behind the approach is to consider this problem as a multi-class classification (having as many classes as maintenance teams). The study focuses on the comparison of different classification approaches for carrying out this task, specifically vector space model, bayesian model, support-vector machine, classification trees and *k*-nearest neighbour classification.
- Cubranic (2004) treats the problem of assigning developers to bugs as an instance of classification. More specifically, it is a multi-class, single-label classification problem: each developer corresponds to a single class, and each document (i..e, a bug) is assigned to only one class (i.e., a developer working on the project). The classification algorithm is based on bayesian learning.
- Anvik (2006), unlike the previous approaches, presents a semi-automated approach for the assignment of reports to developers. The approach applies a machine learning algorithm (namely, support-vector machines) to the bug repository to learn the kinds of reports each developer resolves. When a new report arrives, the classifier produced by the machine learning technique suggests a small number of developers suitable to resolve the report. Finally, the triager (i.e., the person doing the triage) must select the actual developer from the recommended set to whom the bug will be assigned. The triager may make this choice based on knowledge other than that available in the bug repository, such as the workloads of the developers, or who is on vacation.

2.1.6 Context-awareness in recommender systems

Context-aware recommender systems (CARS) generate more relevant recommendations by adapting them to the specific contextual situation of the user (Adomavicius 2011).

In typical recommender research, the focus is on techniques that recommend various products or services based on specific knowledge about the user. Such knowledge includes individual tastes, preferences as well as users' past behaviour (e.g., previous purchases). Several existing approaches focus on recommending the most relevant items to users and do not take into account any additional contextual information, such as time, position, weather, user's mood, presence of other people, or even the type of device which the user is currently using. The traditional view on RS deals with only two types of entities, *users* and *items*, but does not put them into *context* when providing recommendations.

The relevance of contextual information has been recognized by researchers and industry in many different areas (for instance marketing, mobile computing, e-commerce, information retrieval). Besides, context-awareness has been increasingly recognized as a critical issue in many recommendation applications and the importance of this topic is also supported by various papers on CARS in major conferences, such as ACM Recommender Systems.

Several different algorithmic approaches exist on how to deal with context-awareness in RS. Below we describe some approaches, which can be of relevance for OpenReq:

- Baltrunas (2014) presents an algorithmic approach that uses *item splitting* pre-filtering methods for context-aware recommendations. The main idea of "item splitting" is that:

  a) the same recommendable item can be experienced or consumed by users in different contextual settings (for example in summer or in winter), and

  b) the explicit user ratings (preferences of the user) for the items may depend on this contextual setting.

  The approach proposes to split the actual item into two or even more fictitious items, one for each contextual condition. When generating the recommendations the user's current contextual condition is taken into account and a rating prediction for the fictitious item that matches the user's current context is calculated. An evaluation of the item splitting approach shows that it can help to improve the prediction quality of a recommender system in the analysed application scenarios.

- Campos (2014) focuses on *time*, as one of the most valuable contextual factors in many RS domains. The empirical review of (Campos 2014) includes common evaluation practices and present methodological issues related to the comparative evaluation of time-aware RS based on historical data sets. The main point of the work is that the choice of the evaluation conditions impacts the ranking of different recommendation strategies, which can also be observed in context-unaware RS. To counteract this kind of issues, a set of guidelines as well as a corresponding methodological framework for a robust and fair evaluation process is proposed.

- Hussein (2014) focuses on the systems engineering perspective. It introduces a software framework for the development of context-aware and hybrid recommender systems. Despite the high practical relevance of RS in industry, little research on engineering aspects of such systems has been done, and no comprehensive software framework is yet available that supports component-based development approaches for such complex systems. The main challenges when designing such a framework are:

  a) the choice of the appropriate level of abstraction such that the individual components can be (re)used in various application domains, and

  b) the number of possible data sources that need to be integrated.

  The work points out that one of the most important requirements for the design of such a framework is how to acquire and process various types of contextual information. Furthermore, it presents the general architecture, as well as the functionality of the pre-implemented framework components and presents the first analytical and empirical evaluation of the framework.

2.2 Improvements of requirements quality

This section summarizes a *systematic mapping study* (SMS) on the topic of requirement quality improvements performed following the guidelines of (Petersen 2009).

We obtained a first set of references for the SMS performing a manual search. This manual search consisted on reading through the titles, keywords, and when in doubt abstracts of the manuscripts published in the proceedings of the 2015-2017 editions of the two most relevant RE venues (*IEEE International Requirements Engineering Conference, RE*[5]; *Working Conference on Requirements Engineering Foundation for Software Quality, REFSQ*[6]) and in the issues of the *Requirement Engineering Journal*[7] for the same period. We then performed an automatic search, based on digital libraries *(IEEE Xplore*[8]*, ACM Digital Library*[9]*, SpringerLink*[10] and *ScienceDirect*[11]), for the following terms *requirement engineering*, *quality assurance*, *recommender system*, and *quality property* in title, abstract, and keywords. From this initial set of references, we removed the ones not in scope with the topic of requirements quality (e.g., publications in other fields), and non-English publications. We then performed a single iteration of forward and backward snowballing on all the remaining references using *Google Scholar*[12] *cited by* functionality and the paper reference list, respectively. After reading the titles and abstract of the retrieved set of papers (~1200), we selected 163 publications reporting studies deemed relevant for requirements quality improvement in the context of OpenReq.

In Table 2, we show a representative sample of the publications found. By "representative", we mean that the sample has been chosen to cover the requirement quality improvement functionality which the OpenReq infrastructure will offer.

The SMS shows that the main focus of quality improvement deals with structural properties of the requirements, such as natural language text, whereas secondary foci are stakeholder engagement, requirements completeness, and conformance to requirement templates. Therefore, we report the state-of-the-art according to the following quality targets (Figure 2): unambiguity, conformance to templates, completeness, stakeholder engagement, and other less common aspects, each one being presented in the subsections below.

2.2.1 Unambiguity

Ambiguity exists within *natural language* (NL) requirements because NL has a tendency to be ambiguous (Wilmink 2017). When it comes to writing requirements with more clarity (less ambiguity), the first two major steps are the definition and identification of the ambiguities that exist within NL requirements (Berry 2003, 2004). These two steps are:

[5] http://www.re2017.org/
[6] https://refsq.org/2017/welcome/
[7] https://link.springer.com/journal/766
[8] http://ieeexplore.ieee.org/Xplore/home.jsp
[9] https://dl.acm.org/
[10] https://link.springer.com/
[11] https://www.sciencedirect.com/
[12] https://scholar.google.es/

1. Learn to write less ambiguously and less imprecisely.
2. Learn to detect ambiguity and imprecision.

The state-of-the-art in improving requirements quality by addressing ambiguity in requirements documents is largely focused in the second objective, by automatically detecting ambiguity. However, there is still ongoing work to address the definition and framing of ambiguity within requirements.

Table 2. Improvements of requirements quality: state-of-the-art summary

Reference	*Quality target*	*Approach*	*Tool availability*
(Berry 2003)	Unambiguity	Literature Review	No
(Berry 2004)	Unambiguity	Literature Review	No
(Ferrari 2016)	Unambiguity	Stakeholder Interviews	No
(Ferrari 2017)	Unambiguity	NLP (word vectors)	No
(Gleich 2010)	Unambiguity	NLP (POS tagging) Lookup dictionary	Yes
(Huertas 2011)	Unambiguity	Formal language definition	No
(Sabriye 2017)	Unambiguity	NLP (POS tagging)	No
(Wilmink 2017)	Unambiguity	Lookup Dictionaries	Yes
(Yang 2010)	Unambiguity	NLP (several techniques)	Yes
(Arora 2014)	Conformance to requirement template (Rupp)	NLP (text chunking)	Yes
(Arora 2015)	Conformance to requirement template (EARS)	NLP (text chunking)	Yes
(Lucassen 2015)	Conformance to requirement template (user story)	NLP (several techniques)	Yes
(Beaumer 2016)	Completeness	Semantic role labelling Information retrieval	No
(De Vries 2016)	Completeness	Static analysis Evolutionary computing	No
(De Vries 2017)	Completeness	Static analysis Evolutionary computing	No
(Eckhardt 2016)	Completeness	Sentence patterns	No
(Ott 2013)	Completeness	Support vector machines	Yes
(Dalpiaz 2017)	Stakeholder engagement	Gamification	Yes
(Lombriser 2015)	Stakeholder engagement	Gamification	Yes
(Ninaus 2014)	Stakeholder engagement	Traffic light	No
(del Sagrado 2017)	Others (Stability)	Bayesian network	Yes
(Unterkalmsteiner 2017)	Others (General)	Cognitive load theory	No

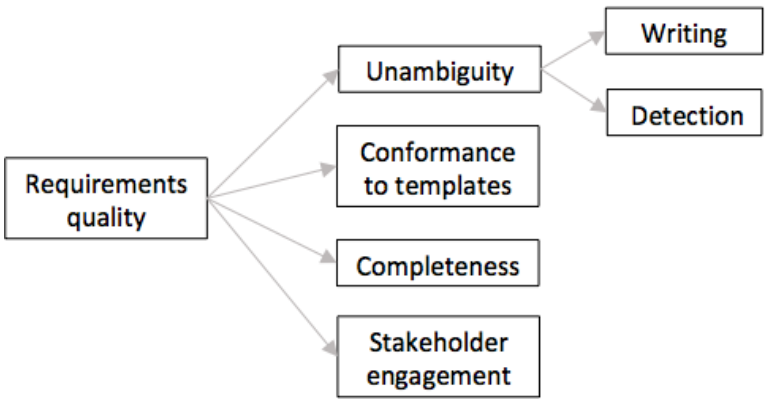

Figure 2. Requirements quality improvements targets

This section begins with addressing the main categories of ambiguity in RE as defined by Berry (2004), followed by the most recent work of (Ferrari 2016) to account for the recent progress in expanding the definition and scope of ambiguity in RE. This section will end with a summary of the most recent work in addressing the second objective above (i.e., learning to — automatically — detect ambiguity in requirements documents).

Berry (2004) collects and summarises past work addressing ambiguity in RE. Most notably, this work lays out a set of linguistic definitions for ambiguity in RE that are subsequently cited and used as starting point for most of the papers to follow in this section. They distinguish between four broad classes of linguistic ambiguity:

- *Lexical ambiguity*, which refers to situations in which a word has several meanings.
- *Syntactic ambiguity*, which occurs when a sentence (or sub-structure of a sentence) has more than one grammatical structure (more than one parse tree, in NLP terms).
- *Semantic ambiguity*, which refers to a situation in which there are no lexical or syntactic ambiguities but still there is more than one meaning to a sentence.
- *Pragmatic ambiguity*, which refers to a sentence that has several meanings depending on the context in which it is used. Whereas semantic ambiguities have more than one meaning even with the surrounding context known, pragmatic ambiguity can be resolved by adding context around the sentence.

These definitions form the basis for the majority of the work on ambiguity in RE.

Ferrari (2016) looks to expand the scope of addressing ambiguity within RE to include the activity of requirements elicitation, a process that is usually conducted verbally. The work begins by reviewing the definitions summarised in the work of (Berry 2003, 2004), followed by adding to those definitions an expansive list of ambiguous situations that were encountered during the research. The novel addition of this work to the knowledge of ambiguity in RE, beyond the experiments with industry practitioners, is the incorporation of *acceptability* from the analyst's perspective. It accounts for interpretability (i.e., the ability to assign at least one meaning to a customer statement), and also whether or not the requirement defined by the customer is reasonable (i.e., a changing definition depending on the analyst). The work includes 12 different situations that were seen during the experiments conducted with practitioners. Out of them, five possible situations based on logical combinations of interpretability and acceptability are proposed:

1. *Interpretation unclarity* involves the situation in which the analyst cannot interpret what the customer is saying, which automatically means that the analyst cannot accept what is being said.
2. *Acceptance unclarity* is when the analyst correctly interprets what is being said, but does not accept it.
3. *Multiple understanding* is the classic ambiguous situation in which the analyst accepts (and therefore interprets) multiple meanings to something a customer said.
4. *Detected incorrect disambiguation* is the scenario that most of the research in RE ambiguity is trying to create: the analyst interprets what the customer is saying, recognises that it is wrong, and does not accept it.

5. *Undetected incorrect disambiguation* is the scenario that RE ambiguity research aims to eliminate: the analyst incorrectly interprets the customer and then accepts it, despite the misunderstanding.

These definitions, although not widespread yet, provide an in-depth perspective into requirements elicitation and the ambiguous situations that can arise.

With the work of (Berry 2003, 2004) and (Ferrari 2016) laying the groundwork for step one of addressing ambiguity within requirements (learning and defining ambiguity within requirements), the next step of automatically detecting and addressing ambiguity within requirements is addressed in the following research. The papers presented here are a mix of look-up dictionaries (i.e., ontologies, and NLP techniques such as parts-of-speech tagging and word vectors).

Gleich (2010) proposes an approach for automatic ambiguity detection through the use of text-matching with regular expressions, similar to the functionality of the Unix command *grep*. The types of ambiguities identified in this work originated from the work (Berry 2003) and a *Siemens-internal guidelines for requirements writing style guide* (no source is included in the work). From these two sources, it implements regular expressions to automatically identify 38 types of ambiguities. Following the creation of the tool, the implementation is tested through a comparison with human annotators. For the study, 50 German and 50 English sentences are extracted randomly from real requirements documents. Then 11 subjects (requirements engineers, masters students, and PhD students) mark ambiguities in these sentences. The results are similar for both English and German at ~95% precision and 86% recall. However, these results are dependent on comparing directly with the abilities of the human annotators. In other words, the results only considered the situations in which the human annotator marked ambiguities. The true precisions and recalls of this tool (for both English and German), when taking into account all possible ambiguities to be marked, are ~40% and ~54%, respectively. Given that it is common to compare an automated tool to that of a human annotator (since manually annotating is the alternative), it is reasonable to consider the first set of results.

Yang (2010) explains the implementation of a tool called *Nocuous Ambiguity Identification* (NAI) to automatically identify nocuous (i.e., harmful) coordination ambiguities in requirements documents. As Engelhardt (2010) explains, "coordination is a procedure that links two sentence elements called conjuncts" (e.g., the use of *and* and *or*). The first stage of the tool (called ambiguity detection) identifies ambiguities through part-of-speech tagging, shallow parsing, word co-occurrence, and word distribution. The second (nocuous) stage of the tool utilises machine learning and manual annotation of ambiguous requirements as harmful (or not) to build a classifier capable of automatically identify ambiguous requirements beyond a nocuous threshold. The results of the work show the tool is able to produce precision and recall as high as 80% in identifying harmful ambiguous requirements.

Huertas (2011) proposes a formal language to measure the degree of ambiguity in a document based on the knowledge of the stakeholders who will be involved in the creation of the document. The research is predicated on the work of (Swinney 1979), which shows that "even when a context is strongly biased indicating a single meaning for a word, reader's brain accesses every possible meaning for that word". Huertas (2011) extends this finding in the context of RE to imply that if all stakeholders have a shared knowledge state about the words and phrases used in the requirements document, then "even if the specification is ambiguous in many ways, it will be completely unambiguous and harmless to that given set of stakeholders". The presented framework is used in a case study with six stakeholders from a

company. The results show that out of the 20 requirements analysed, only eight words are "potentially ambiguous". These results show that the framework can be applied with real practitioners, but does not test the validity of the framework itself, nor questions the assumptions made about stakeholder's shared-knowledge assumed from the work of Swinney.

Wilmink (2017) proposes the use of weak and strong phrases as well as subjective and non-verifiable terms as a process of identifying ambiguities in requirements documents. As described by (Wilson 1997), "weak phrases is the category of clauses that are apt to cause uncertainty and leave room for multiple interpretations", whereas the absence of those weak clauses means the phrase is strong. Subjective terms are "words of which the semantics [are] not objective" such as *user friendly* and *easy to use*, whereas non-verifiable terms are "hard to verify as they offer a choice of possibilities" such as *provide support* and *as a minimum* (Femmer 2014). Wilmink (2017) develops a tool called *Tactile Check* that implements the lookup functionality available from two independent tools, *NASA ARM* (Wilson 1997) for weak and strong phrases and *SMELL* (Femmer 2014) for subjective and non-verifiable terms. Although the tools have more complex functions, the work harness just the basic lookup functionality. To test the viability of the tool, Tactile Check is applied to two requirements documents from industry, analysing a total of 293 requirements and creating a total of 454 annotated phrases. Then, three analysts review the annotations created by the tool to summarise their perspective on the usefulness of the tool. The results show that the weak phrases are beneficial to identifying ambiguous requirements with precision at 92% and recall at 87%, but the strong phrases are not. It appears that the usefulness of subjective and non-verifiable terms is not tested directly, but rather used to strengthen the dictionaries used for weak and strong phrases.

Sabriye (2017) proposes using part-of-speech tagging to automatically identify ambiguities in requirements documents. In particular, it looks for four identifiable patterns in the POS tagging: lack of a full sentence stop, passive voice, more than one parse tree, and the existence of *and* and *or* logic within the same sentence. These four rules are designed to cover a number of ambiguity categories defined by (Berry 2003). This research is preliminary as the methods described are not implemented, although it includes as plans for future work the implementation and testing.

Ferrari (2017) proposes a system to estimate the degree of ambiguity in "typical *Computer Science* (CS) words". The system creates context-dependent word vectors using Wikipedia articles under certain categories as the basis for how the word vectors are defined. It utilises the word2vec algorithm (Mikolov 2013) to create different sets of word vectors for the same bank of words, using a different category of Wikipedia articles for each set. With this technique in place, the authors can create different word vectors for a word such as *system* by drawing on Wikipedia articles first from the CS domain, and then from the medical domain. With this technique at hand, multiple sets of word vectors are created for the top 100 CS words (top 100 words used in a randomly created corpus from Wikipedia CS articles), comparing the difference in word vectors (difference in meaning) to the domains of Electronic Engineering, Mechanical Engineering Medicine, Literature, and Sports. The results show that many of the most used — and arguably most important — words such as *code*, *database*, *support*, and *programming* have different meanings across domains. This points to the potential ambiguity contained within these words when working across domains.

2.2.2 Conformance to templates

Templates or boilerplates are frequently used to structure requirements. This allows the authors of the requirement documents to focus on eliciting the specific information requested by the template. Moreover, such structure makes automatic requirements analysis easier. Therefore, it is important, from a quality assurance perspective that the requirements conform to the templates.

Arora (2013, 2015) proposes a template conformance checking technique based on text chunking — a NLP approach which identifies sentence segments (i.e., chunks). The templates under investigation, namely *Rupp* and *EARS*, are transformed into backus-naur form grammars. This allows to define a series of pattern-matching rules for checking conformance based on the identified chunks. The tool *RETA* (REquirement Template Analyzer) is developed to demonstrate the proposed approach. The evaluation consisted of four case studies (two using Rupp and two using EARS) in the context of safety critical systems (e.g., satellite control, nuclear energy safety control) covering more than 1700 requirements. The results show that, from a practical standpoint, the tool is accurate for automatically checking requirements conformance to templates. Specifically, the number of false positives and false negatives is small across all cases in absolute numbers (38 in total) and as percentage of total number of non-conformant requirements (from 5% to 18%).

Lucassen (2015) presents an NLP-based approach to assist stakeholders in producing high quality requirements by enforcing the conformance to user stories templates. The work first devises a conceptual model of high-quality user stories based on three high-level criteria — syntactic quality, semantic quality, and pragmatic quality— which are further broken down into 14 criteria (e.g., independent, uniform, unique, and complete). A tool, *AQUSA*, enforces that the guidelines for writing user stories, based on these criteria, are correctly followed. The tool supports syntactic criteria and, partially, pragmatic criteria, whereas semantic ones are not covered. The evaluation of the tool is done on three sets of user stories collected from industry. The tool shows 71% precision in detecting the user stories that violated quality criteria. The results show that one of the most common violations is minimality —i.e., the syntactic quality property for which a user story should not contain more than *role* (i.e., a relevant role should always be defined), *means* (i.e., define a subject with an intent targeting an object), and *ends* (i.e., a reason for the mean).

2.2.3 Completeness

A requirement is complete if it contains the necessary information to enable proper implementation and verification. Therefore, a complete requirement expresses the whole need and states all its conditions and constraints.

De Vries (2016, 2017) proposes an approach based on symbolic analysis and evolutionary computation which identifies a set of counterexamples representing requirement incompleteness in the context of hierarchical requirements modelling. In the proposed approach, a requirement is considered to be incomplete if there exists a case where a parent requirement is unsatisfied while the set of its decomposed requirements are satisfied. Thus, the approach is able to identify incomplete requirements decomposition. The work is demonstrated on a set of requirements for a cruise control system. The combination of static analysis and evolutionary computing yields better results than static analysis alone and provides a complete set of counterexamples through the application of evolutionary computing.

Baumer (2016) proposes an approach to identify incompleteness in user stories and suggests specific corrective actions based on the set of existing user stories. The work employs semantic role labelling to assign a semantic label to each predicate and then leverages ontologies to check whether the arguments expected by each semantic label are also present (e.g., the predicate send has arguments such as sender and send-to). Suggestions for filling missing arguments are given based on similar user stories obtained using information-retrieval approaches.

Similarly, Ott (2013) proposes an approach to classify large amount of natural language requirements into topics to help stakeholders recognize incompleteness when reviewing requirements. It proposes a tool, *ReCaRe*, which uses support-vector machines to categorize text. The tool is evaluated in a small case study with ten masters' students on a 3000 pages requirement specification document provided by Mercedes-Benz. The results show the initial benefits of using topic classification for the identification of incomplete specification. However, the tool does not automatically recommend improvement actions.

Rather than on functional requirements, the approach proposed in (Eckhardt 2016) focuses on incompleteness, and related corrective actions, for performance requirements. It provides a definition of completeness for performance requirements based on the presence of content elements in the textual representation of the requirement. Accordingly, it defines i) strongly complete, ii) weakly complete, and iii) incomplete requirements if i) all mandatory elements are explicitly defined in the text, if ii) all mandatory elements are explicitly or implicitly defined in the text, or if iii) at least one mandatory element is missing. Based on this definition, sentence patterns are derived, which correspond to the three levels of (in)completeness. The evaluation is based on 58 performance requirements gathered from industrial specifications. The results show that 86% of the existing requirements can be rewritten following the suggested patterns. However, only 18% of the requirements are strongly complete, whereas 32% are weakly complete. The remaining incomplete ones are not testable and will likely cause problems in later phases of the life cycle.

2.2.4 Stakeholder engagement

Stakeholder engagement deals with the participation of stakeholders in requirements activities, such as workshops and meetings. Lack of engagement can lead to poor quality requirements (e.g., requirements that do not take into account all perspectives) and ultimately to project failure.

Gamification techniques are shown to improve requirements quality by fostering active participation of stakeholders. Lombriser (2015) uses the most popular gamification elements (i.e., points, badges, and leader boards) in the context of scenario-based RE. According to the proposed framework, gamification has a positive impact on stakeholders engagement because it improves their motivation, which, in turn, positively impacts performance measured in terms of creativity, productivity and requirements quality measured using the *INVEST* model, according to which a user story should be *I*ndependent from others, *N*egotiable, *V*aluable, *E*stimable, *S*mall enough to fit in an iteration, and *T*estable. To test the framework, a web-based requirements engineering platform (called *Captain Up*) is developed, which contains 17 different game elements. The experiment was carried out with 12 IT professionals and evaluated the performance of the stakeholders using the platform with respect to the ones who did not. The results show that gamification improves quality characteristics of requirements (user stories) such as independence, effort estimations, size, and testability. However, no significant results are obtained for characteristics such as negotiability and value. Moreover,

using the *Kano* model, which can determine how satisfied or dissatisfied an end user would be given not only the presence but also the absence of certain features, more than half of the user stories produced by the stakeholders using the gamified platform were rated as attractive requirements.

Dalpiaz (2017) presents a gamification-based approach to foster the involvement of the crowd in requirements engineering tasks. Using the design science methodology approach (Peffers 2007) —i.e., tightly interpolating cycles of artefact development and evaluation— the platform *REfine* is devised based on a series of interviews with ten expert requirement engineers. On the platform, the crowd of stakeholders shares their needs and learns from each other to reach consensus on the requirements to be implemented. This is achieved through a mechanism of comments and branching/merging which enables explicit negotiation. Nevertheless, the crowd should be engaged to ensure their participation on the platform. To that end, several gamification elements are employed, such as leader board, points, roles, and explicit endorsement. The platform is evaluated with a one-month long study involving 19 participants (i.e., product managers, developers, clients, end-users). The results show that the motivation to perform requirements engineering task was improved and that basic features, such as commenting and voting, were considered very useful as opposed to pure game-like features (e.g., leaderboard) which were rated neutrally. Nevertheless, a group of experts evaluated the output of the requirements engineering activities performed on the platform and suggested that the quality of the requirements is not significantly better than usual (e.g., using traditional platforms). The work also points out the difficulty in involving a large number of participants as the crowd might not be representative of the domain of the system, or the vocabulary can vary between such large population of different stakeholders.

Ninaus (2014) presents a tool, *IntelliReq*, which leverages a traffic lights feedback mechanism to engage stakeholders in the different requirements-related tasks, including quality assurance. For example, a red traffic light is associated to a neglected requirement indicating that it needs to be reviewed. From a psychological standpoint, this technique leverages the stakeholders' needs for closure with a task and persuades them to take action until a state of completeness is reached (i.e., green light). A qualitative assessment of the traffic light feature with 20 subjects shows that such quality assurance recommendations are helpful and should be constantly displayed. A controlled experiment with 32 subjects (i.e., computer science students) acting as release managers working on an existing requirement model shows that the group using traffic light recommendation needed less interaction steps and less time to successfully complete a release with respect to a control group (i.e., subjects not supported by the traffic light).

2.2.5 Others

The results of our SMS shows that there are other requirements quality characteristics which are less investigated.

Unterkalmsteiner (2017) proposes a taxonomy as the basis for a requirements quality framework. Through a series of interviews with stakeholders of the Swedish Transportation Administration, the work prioritizes a set of quality attributes and defines the relationships among these attributes. The novelty in the approach lies in the use of cognitive load theory for making decision regarding the rules to be followed when writing requirements specification. The work suggests several heuristics to measure cognitive load indirectly as well as directly. Although it is at an early stage, the inclusion of cognitive measures for requirements quality recommendation appears to be promising.

A recent work of (del Sagrado 2017) aims at recommending requirements which are likely to change, therefore, suggesting to stakeholder whether a requirement specification is sufficiently accurate or might require further revision. Using a bayesian network —built based on data from longitudinal interviews with two expert requirement engineers— the work shows that it is possible to locate which requirement specification concerns should be addressed to improve its stability.

3 PERSONAL STAKEHOLDERS' RECOMMENDER ENGINE

In this section we give an overview of the recommendations for the stakeholder's approach that will be developed in OpenReq. Specifically, subsection 3.1 describes the recommendation tasks that are part of the stakeholders' recommender engine, subsection 3.2 details the algorithms, evaluation and technologies that will be used, and subsection 3.3 provides a description of the architecture of the stakeholders' recommender engine. Figure 3 summarizes the main points that will be further explained in the next subsections.

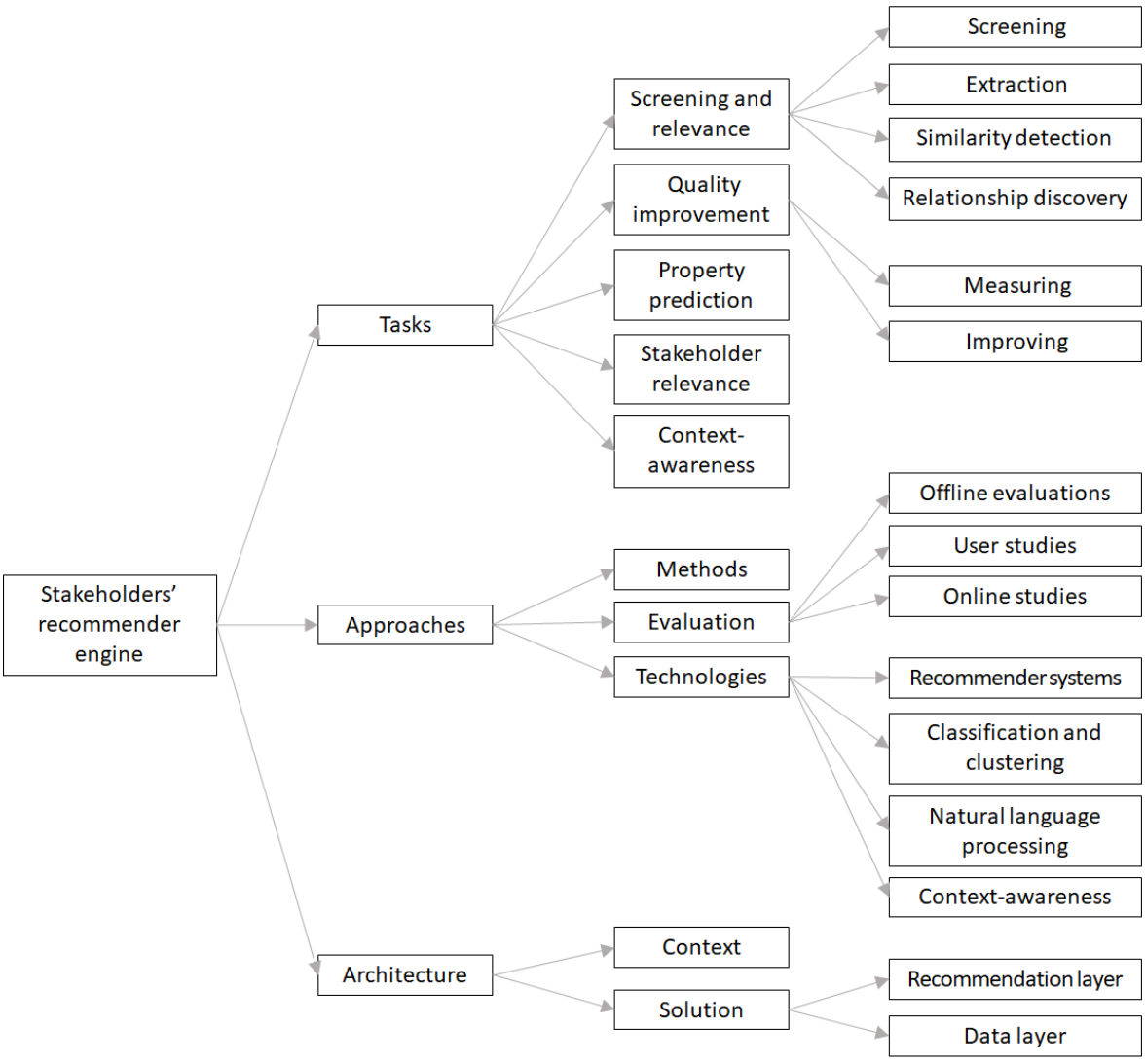

Figure 3. Personal recommendations in OpenReq overview

3.1 Recommendation tasks

As described in the DoA, recommendations for individual stakeholders will be related to the screening and recommendation of relevant requirements, to the improvement of requirements quality, to the prediction of requirements properties, and to the identification of relevant stakeholders. These recommendations will be context-aware, meaning that the current context of the stakeholders (e.g., historical data about the roles the stakeholder had in the past, past involvements in specific requirements, and the last requirements the stakeholder has

consulted) will be taken into account when providing the recommendations. In the following, we describe each one of these groups of recommendations more precisely:

A. Recommendations about screening and relevant requirements. The recommendations in this group are related to the:

1. *Screening of requirements.* To reduce the overall efforts in requirements engineering tasks, a systematic screening of the available and relevant requirements is essential. This screening can include, among others, requirements of the current project (for instance from past releases) as well as requirements from other projects. Such techniques (for instance, reuse appropriate requirements from other projects) are the basis for saving a lot of effort when adding requirements to a certain project. In addition, screening of requirements could be used for identifying outdated requirements (i.e., requirements that are old and are not relevant any more for a software project) that could be eventually removed to maintain requirements up to date. This is especially the case in large open source projects that are maintained for years, where some old requirements have not been implemented and will never be implemented (because they are not relevant anymore).
2. *Extraction of actual requirements*. In most industrial projects the requirements have to be extracted out of an integrated document (for instance, a tender document). In order to reduce the effort of the requirements engineers the idea is to automatically recognize and recommend relevant requirements out of such documents.
3. *Identification of similar requirements*, in the same project or from previous ones. In RE processes, major problems occur very often in cases where different people are responsible for the identification of requirements (especially out of text documents). There exist often duplicate requirements which are formulated differently by different people but which describe the same issue. In such cases an intelligent identification of similar requirements, both in the same project as well as in different projects, adds a significant benefit to state-of-the-art requirements engineering tools.
4. *Identification of related requirements* in the same project. Projects usually consist out of a large number of requirements. The identification of related requirements in those large sets is a key feature for different real world scenarios. One famous scenarios is, for instance, release planning, where it is essential to know which requirements are related to other requirements. In this case, relatedness could be identified, for instance, when two requirements are mentioning the same functionality.

B. Recommendations about improving the quality of requirements. In this case, the recommendations are related to:

1. *Measuring the quality of requirements* to detect bad quality requirements. Given the state-of-the-art presented in Section 2.2, we will focus on measuring quality aspects related to: i) textual properties of the requirements (e.g., ambiguity, incompleteness, redundancy), ii) structural properties of the requirements (e.g., conformance to template), and iii) stakeholder-related properties (e.g., engagement and motivation).
2. *Improving the quality of requirements*. For instance, by offering suggestions about rewording and standardized glossary to remove ambiguities and fill-in incomplete information, corrective actions to structure the requirements according to templates (if in use), and fostering stakeholders engagement through, for example, gamification elements.

To accomplish both points we will use information available from the current and past projects, such as the textual representation of the requirements, their history, the template used (if any), and stakeholders preferences.

C. Recommendations about requirement properties. The focus here is to predict properties of requirements such as priority. For that matter, we will use the information available (i.e., the requirement per se plus the values assigned to its properties) in past and current projects about similar requirements to the one we want to recommend the requirement properties.

D. Recommendations about relevant stakeholders. Here, the recommendations aim at detecting stakeholders who can cooperate on the definition of requirements. We differentiate here two different scenarios: first, supporting the recommendation of stakeholders relevant for a new software project (taking into account for this assignment the stakeholders' workload or the project domain, for instance), and second, supporting the recommendation of stakeholders for a specific requirement (taking into account the requirements in which stakeholders participated in past projects).

E. Context-aware recommendations. We aim to determine whether pull recommendations (stakeholders trigger them when needed) or push recommendations (automatically delivered to stakeholders) can be applied in a specific context. In general, context-aware recommendations are based on additional information about, e.g., the stakeholder or the project itself. In the case of the stakeholder, for instance, context information can be described as the roles the stakeholder had in the past, the participation duration during each project, past involvements of specific requirements, the skills, etc. In addition, based on historical data, stakeholders might be proactively recommended (push) for specific tasks, or a specific stakeholder pulls requirements that might fit.

3.2 Approach, algorithms, evaluation and technologies

In the following we describe the approach(es) (including algorithms) that will be used in each one of the tasks introduced in subsection 3.1. Afterwards, the technologies used for the stakeholders' recommendations approach are presented.

3.2.1 Approach and algorithms

The OpenReq approach will achieve each of the personal recommendation tasks, presented in Figure 4, as follows:

A. Recommendations about screening and relevant requirements

1. *Screening of requirements*. We will develop and evaluate content-based as well as collaborative recommendation approaches while taking into account available requirement metadata, such as for instance:
    a) Title of the requirement,
    b) Description of the requirement,
    c) Tags (user entered tags as well as automatic generated tags),
    d) Ratings from other stakeholders (in group settings), and
    e) Postings.

    The developed algorithms will be evaluated on predictive quality (for instance how well the algorithms predict relevant requirements and related artefacts in the current

project context). Algorithm development will apply and extend existing recommendation libraries such as Apache Mahout[13].

2. *Extraction of actual requirements*. From the technical perspective, a binary classifier (Manning 2014) will be used in combination with some basic Natural Language Processing (NLP) techniques (Winkler 2016) (to identify common words in requirements such as "must", "have/has to", etc.). For this we envision to use several NLP techniques such as stopword removal, stemming, lemmatization, tense detection, and bigrams detection.

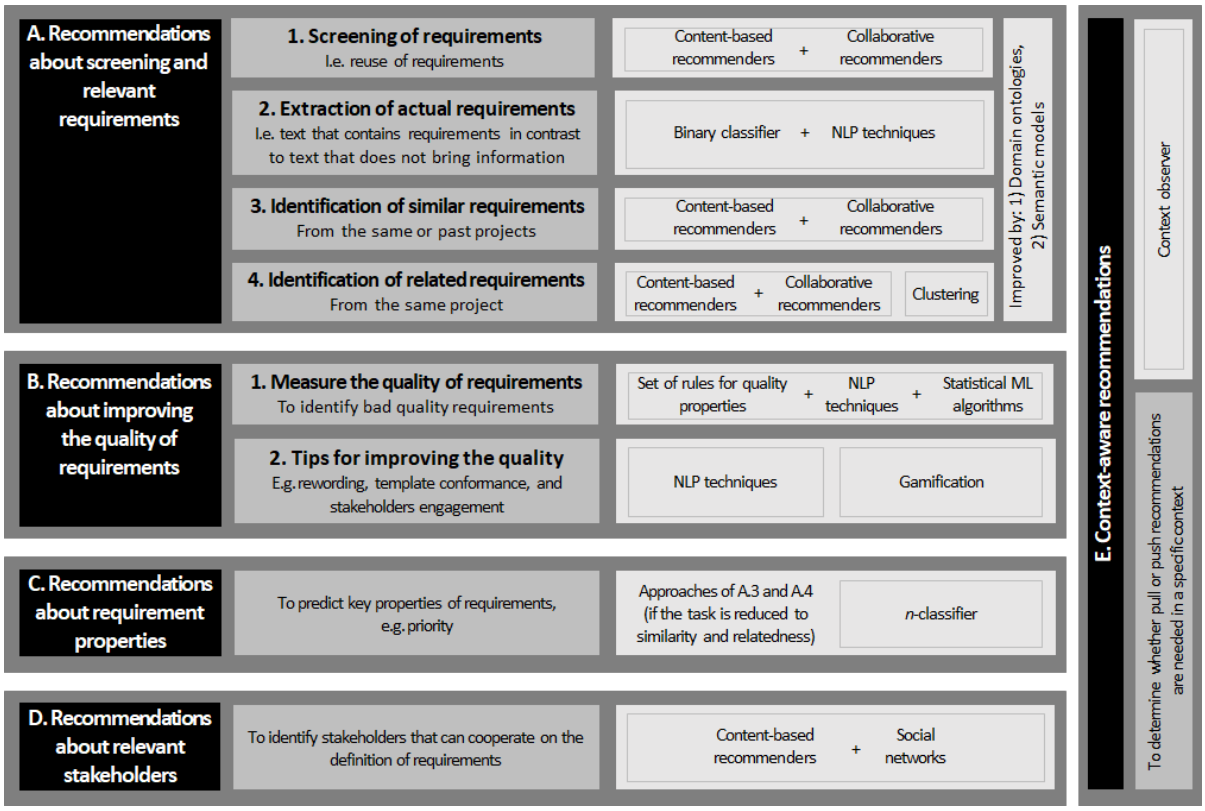

Figure 4. Algorithms for personal recommendations in OpenReq

3. *Identification of similar requirements*. We will use a hybrid solution which makes use of the advantages of content-based and collaborative recommender systems as well as clustering learning techniques. The content-based recommender system will use tags, title, description and postings of a requirement and the collaborative filtering recommender system will use ratings from other stakeholders as input source. The objective of the hybrid system is then to identify identical requirements in order to remove duplicates as well as to detect similar requirements which can then be reused in other projects. Furthermore, posting data (e.g., user comments of requirements) and assessment data of requirements can also be taken into account in order to improve the overall performance of the used approaches and the evaluation quality. We do not

[13] http://mahout.apache.org

discard to combine this approach with the similarity detection that is being carried out in T5.3, where similarity among requirements is identified from an NLP point of view (for more information see deliverable D5.1).

4. *Identification of related requirements*. For this purpose, different approaches identified in the state-of-the-art will be investigated. In the first approach, content-based (based on semantic and text-based similarities) and collaborative recommendation approaches (based on context information) will be used in a hybrid recommender taking into account the available requirement metadata. The second approach is based on topic modelling (Chien 2016), which can be used to associate a label / tag (i.e., a topic) to a requirement or a subset of them, and then cluster the requirements in groups of related ones. The clustering can be done at different level of granularities (e.g., a hierarchy of topics), achieving different levels of relatedness.

   The previous identification task can be improved by the use of: a) domain ontologies, especially to identify synonyms that are domain-specific, and b) semantic models, to specify how requirements are semantically related among each other.

B. Recommendations about improving the quality of requirements

1. *Measuring the quality of requirements.* A set of rules reflecting quality properties will be identified from existing related work (see Section 3.1) and adapted to OpenReq. These rules can then be used against requirements to check their quality, compounding them to calculate a quality score. The majority of the approaches to identify quality concerns leverage raw textual data (i.e., the requirement text itself). Therefore, NLP techniques will be heavily utilized. Among those, text chunking (Arora 2015) and part-of-speech tagging (Arora 2014) will be used to identify basic units of text. Later, the units will be analysed to find, for example, usage of passive voice (considered a bad quality indicator) (Femmer 2014), or missing information (e.g., an active verb missing a direct object) (Baumer 2016). Another possible use of NLP is the resolution of anaphora in the requirements text (Ferrari 2016) —i.e., identify references to the same entity, but using different names, within the text. Similarly, NLP and statistical machine learning algorithms (such as SVM and bayesian network) (del Sagrado 2017) (Ott 2013) can be used to check conformance to requirements template. In addition, the quality of the requirements can be further analysed through personal ratings of stakeholders via several properties of the requirements, which could be used to improve stakeholders engagement and motivation. However, this special aspect of quality will be also dealt with in WP4.

2. *Improving the quality of requirements*. Simple word lists and thesauruses, together with NLP techniques can be used to build recommenders that can point out improvement actions on the basis of the previously defined rules. These improvement actions vary accordingly to the complexity of the task. For the case of passive voice a rewording in active voice can be done automatically (Eckhardt 2016) (e.g., for the requirement in passive voice "*Users should be alerted by the system when new documents are assigned to them.*", the following rewording will be automatically proposed "*The system should alert users when new documents are assigned to them.*"). In contrast, for other cases (e.g., ambiguities resolution, or incomplete information about a requirement) the recommender can only hint at the presence and location of a quality concern but human intervention is likely to be required (e.g., filling-in information gaps) (Eckhardt 2016). NLP techniques (e.g., text chunking) are used to

suggest how to fill-in missing information in requirements templates (Arora 2015). Finally, as the state-of-the-art shows that stakeholders engagement plays an important role in improving requirements quality, we will use gamification approaches (e.g., points, leaderboard and badges) to improve motivation (Dalpiaz 2015, 2017).

C. Recommendations about requirement properties

The state-of-the-art has lead us to identify several approaches that could be used for this task. One approach is based on matching the requirement at hand (i.e., the requirement for which we want to predict the property) with similar requirements that have already been defined in past or ongoing projects, reducing this task to a similarity and relatedness detection case where approaches presented in points A.3 and A.4 of this section can be used: one based on a hybrid recommender combining collaborative and content-based techniques (possibly improved with similarity NLP techniques, and one based on topic modelling. Alternatively, the recommendation of each requirement property can be seen as a classification task with *n* classes where machine learning approaches can be used to assign a value to the specific property of a requirement (with a similar approach of that used in (Fitzgerald 2011)).

D. Recommendations about relevant stakeholders

The approach is based on using a content-based recommender that, by analysing existing social networks (e.g., typical roles of stakeholders in past software projects), individual strengths of stakeholders (e.g., topics the stakeholder has contributed to and related topics the stakeholder could be interested in) and personal availabilities, will create a user profile that will be used to match requirements to stakeholders (see, for instance, (Mooney 2004) and (Castro-Herrera 2009)). Furthermore, the approach will provide a ranking of recommended stakeholders based on the amount of relevant topics (tags) matched as well as the time aspect. For example if a stakeholder developed three Android applications in the last year, she will be ranked higher than a developer whose last Android project was done five years ago. This content-based recommender can be improved to take into account requirements topics in which stakeholders have interest in, adapting the well-known weighting schemes used in collaborative filtering techniques (Said 2012).

E. Context-aware recommendations

A *context observer* component will be integrated in OpenReq, which will take into account contextual information to decide in a personalised way when, what, and in which way recommendations will be delivered. For instance, we can use the history of the stakeholder activities in using OpenReq to know if she is too busy to receive notifications on tips related to requirements quality.

3.2.2 Evaluation

Evaluation is important in assessing the effectiveness of recommendation algorithms. Three types of evaluations are available in the case of recommender systems (Beel 2013):

- *Offline evaluations* are based on historical data, e.g., a dataset that contains information about how users previously rated movies. The effectiveness of recommendation approaches is then measured based on how well a recommendation approach can predict the users' ratings in the dataset. While a rating is an explicit expression of whether a user liked a movie, such information is not available in all domains, which will be the case of OpenReq. In such cases, offline evaluations may use implicit measures of effectiveness. For instance, in the case of recommending related

requirements, it may be assumed that a RS is effective if it is able to recommend as many requirements as possible that will be considered as related by a human.

- *User studies* involve from a few dozens to hundreds of users, who are confronted with recommendations created by a recommender system. Possible tasks of the users in such settings could be to judge which recommendations fit best or how good the recommendations are. Besides the recommendation accuracy the quality of recommendations can have many further different dimensions like for instance user satisfaction with recommendation (or with a result in case of a decision process) and user acceptance of a recommendation.
- In *online evaluations* (also known as A/B tests), recommendations are shown to typically a large user group (up to thousands of users) of a real product or service. The main goal of online evaluations is to test different recommendation approaches (for example different recommendation heuristics). Therefore, in this case of evaluation, the recommender system randomly picks at least two different recommendation approaches (heuristics) to generate recommendations. The effectiveness is measured with implicit measures of effectiveness such as *conversion rate*[14] or *click-through rate*[15].

When evaluating the accuracy of ratings, the commonly used metrics are the *mean absolute erro*r and *root mean squared error* (Gunawardana 2015). In contrast, when evaluating the usage prediction (i.e., if the recommended items are useful/of interest to the user), metrics from the information retrieval field can be used, such as *precision*, *recall*, *F-measure* or *area under the ROC curve* (Gunawardana 2015). Recently, diversity, novelty, and coverage are also considered important aspects in evaluation (Lathia 2010).

In OpenReq, we will mostly use offline evaluations with user studies, combined with metrics from the information retrieval field. In addition, due to the recently criticism to reproducibility in RS (Beel 2016), we will try to follow some of the advices given in this same publication to make our evaluations as reproducible as possible. This is just a first plan for evaluation, and it will probably evolve as the project progresses.

3.2.3 Technologies

The stakeholders' recommendations will be based on a microservice architecture offering and consuming RESTful services (see subsection 3.3). With that aim, several general technologies will be used. For a more detailed description of the OpenReq relevant technologies we refer to the deliverable D1.4.

In addition, we will need specific technologies for recommendation systems, classifications and clustering, and NLP, which are described in subsections 3.2.3.1, 3.2.3.2, and 3.2.3.3, respectively. These subsections contain a list of available technologies for the field. The final technologies to be used in OpenReq will be chosen taking into account the type of license of

[14] Conversion rate is the proportion of users of a system who take action to go beyond a casual content view or website visit, as a result of subtle or direct requests from marketers, advertisers, and content creators.

[15] Click-through rate is the ratio of users who explore a specific item (e.g., by clicking a link) to the number of total users who were given the possibility to explore the item.

the OpenReq system, so they are compliant with it. Finally, subsection 3.2.3.4 includes a summary for technologies used.

3.2.3.1 Technologies for recommender systems

This section describes a tentative list of available packages and tools that could be used to develop the personal recommendations for stakeholders in OpenReq:

- *Apache Mahout*[16] (Apache License v2). This Java library supports a lot of collaborative filtering algorithms. It is highly scalable, but content-based recommendation is not supported by the library. The library is characterized by having a good documentation, high performance, and maintainability.
- *Apache PredictionIO*[17] (Apache License v2). It is a very active open source project that provides a machine learning server that can be used to create a recommender system. It supports both content-based and collaborative filtering. Additionally, it has a general interface to develop other types of recommenders (such as knowledge-based).
- *LibRec*[18] (GNU General Public License). LibRec is a Java based recommendation engine with more than 70 kinds of recommendation algorithms. Specifically, these algorithms can be divided into benchmark algorithms, collaborative filtering algorithms, content-based algorithms, context-aware algorithms, hybrid algorithms and other extended algorithms.
- *LightFM*[19] (Apache License v2). It is an actively-developed Python implementation of a number of collaborative- and content-based learning-to-rank recommender algorithms. It easily scales up to very large datasets on multi-core machines.
- *MyMediaLite*[20] (GNU General Public License). MyMediaLite is, comparable to Apache Mahout, a recommendation library for collaborative filtering that runs on .NET. It contains not that much algorithms than Apache Mahout but it supports two commonly occurring collaborative filtering scenarios: 1) rating prediction (e.g., on a scale of 1 to 5 stars), and 2) item prediction from positive-only feedback (e.g., from clicks, likes, or purchase actions).
- *RankSys*[21] (Mozilla Public License v2). It is Java recommendation system that includes support for the evaluation and enhancement of novelty and diversity.

3.2.3.2 Technologies for classification and clustering

Several frameworks and libraries are already available for classification and clustering. In the following, we introduce the most famous ones:

[16] http://mahout.apache.org/
[17] http://predictionio.incubator.apache.org/index.html
[18] https://www.librec.net/
[19] https://github.com/lyst/lightfm
[20] http://www.mymedialite.net
[21] https://github.com/RankSys/RankSys

- *Apache Mahout*[22] (Apache License v2) includes a Java implementation for popular machine learning algorithms for classification, clustering and topic modelling. The library is highly scalable, and it is characterized by having a good documentation, high performance, and maintainability.
- *Scikit-learn*[23] (BSD license) offers a Python implementation of the most popular machine learning algorithms for classification and clustering. The library is characterized by an excellent documentation, high performance, ease of use and maintainability. The drawbacks are scalability and heavy multiprocessing instead of lightweight multithreading (due to Python GIL limitations - see https://wiki.python.org/moin/GlobalInterpreterLock).
- *Weka*[24] (GNU General Public License), developed in Java, supports diverse machine learning approaches for classification, and clustering and attribute selection. In addition to the graphical user interface provided, it can be accessed via a Java API.

3.2.3.3 Technologies for NLP

In the case of NLP, there are multiple libraries and toolkits that deal with a different variety of techniques. Some of the most popular ones are:

- *Nltk*[25] (Apache License v2), developed in Python, is used for NLP common tasks, such as tagging, tokenizing, and stemming.
- *Stanford CoreNLP*[26] (GNU General Public License v3+) is a popular NLP toolkit for Java. Apart from the common NLP tasks, it supports advanced processing such as named entity recognition, dependency analysis, and part-of-speech tagging. This toolkit is still maintained by the Stanford NLP group.
- *OpenNLP*[27] (Apache License v2) is a Java API that supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, language detection and coreference resolution.
- *Word2Vec*[28] (Apache License v2) is an algorithm developed by (Mikolov 2013) at Google to assign arbitrary-length vectors to words that correspond to the meaning of the word. The vectors created depend on the corpus used, therefore it is important to know which corpus was used when working with a set of word vectors.

It is important to highlight here the fact that OpenReq project will not only deal with English language, but also with Italian and German, since the telecom trial deals with text written in the Italian language and, for the case of Siemens, with text (partially) written in the German language. This diversity of languages used to write the text analyzed is a challenge for the

[22] http://mahout.apache.org/
[23] http://scikit-learn.org/
[24] http://www.cs.waikato.ac.nz/ml/weka/
[25] http://www.nltk.org/
[26] https://stanfordnlp.github.io/CoreNLP/
[27] https://opennlp.apache.org/
[28] https://code.google.com/archive/p/word2vec/

project. The majority of the existing NLP approaches target the English language, as they are trained and validated using English text corpora. Although NLP approaches and software libraries exist for the other two languages (Basili 2015) (Rehbein 2012), their performances (e.g., precision) might be inferior compared to the well-established, English-based ones.

3.2.3.4 Technologies for context-awareness

The technologies used for context-aware recommendations depend on the technologies used for the final platform as we will rely on interaction and history data. The following summarizes technologies that can be used und reviewed during the project:

- *MyLyn*[29]*/Eclipse Usage Data Collector*[30]. An interesting data source for context-aware recommendations is interaction data. Interaction data describes the usage by tracing the events happening in the system (e.g., the user clicks in a UI element, the systems receives a notification). MyLyn and Eclipse are examples of existing tools which can be instrumented to collect interaction data, in this case, in the Eclipse Integrated Development Environment. Data gathered from these tools can help to understand how a system is used and on what tasks a user is performing. Recommendations, based on such context data, can be given, for example, to optimize a task (e.g., complete a task with fewer clicks).

3.2.3.5 Technologies summary

The applied technologies will be based on micro-service architecture. Apart from the general technologies stated in D1.4, the technologies below will be utilized:

- Recommender system framework. At least one of the following will be used (we do not discard to enlarge this list in the future): Apache Mahout, Apache Prediction IO, LibRec, LightFM, MyMediaLite, and RankSys.
- Classification and clustering framework. One of the following will be tentatively used: Apache Mahout, Scikit-learn, and Weka.
- NLP framework. One of the following will be tentatively used: Nltk, Stanford CoreNLP, OpenNLP, and Word2Vec.
- Context-awareness framework. One of the following will be tentatively used: MyLyn, and Eclipse Usage Data Collector.

Generally, we will apply or expand the algorithms and capabilities of these technologies rather than develop entire new ones. In addition, we do not discard that some technology is added to this list, especially in terms of the frameworks used for recommender systems, classification and clustering, NLP, and context-awareness. Finally, we want to highlight that the final choice of technologies will be done taking into account the type of license of the OpenReq system, so the technologies used in this workpackage are compliant with it.

[29] https://www.eclipse.org/mylyn/

[30] https://www.eclipse.org/org/usagedata/

3.3 Architecture

This subsection describes the planned software architecture for the software services that realize the stakeholders' recommendations approach. These services are collectively called *stakeholders' recommender engine*. Subsection 3.3.1 describes the general context of the stakeholders' recommender engine, and subsection 3.3.2 the architecture overview.

3.3.1 Context of the stakeholders' recommender engine

A view of the context in which the stakeholders' recommender engine operates is shown in Figure 5. As can be seen, the engine can be used in different RE tasks.

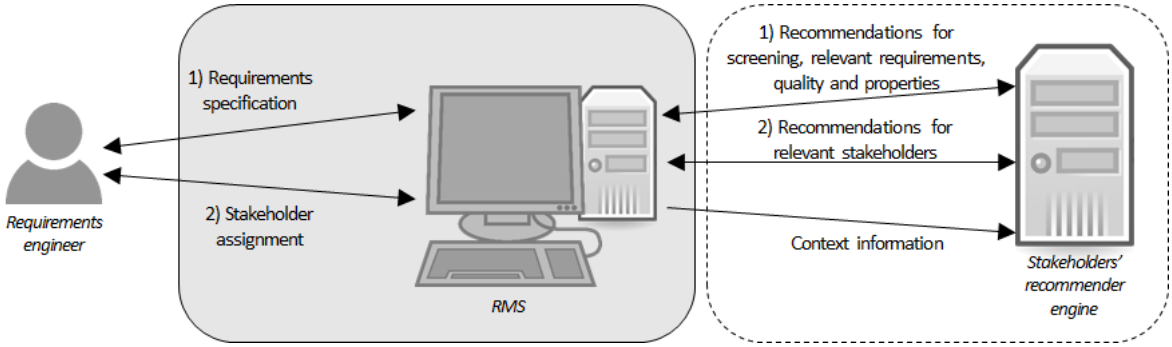

Figure 5. Context of the stakeholders' recommender engine

The stakeholder role that interacts with the stakeholders' recommender engine will be usually a *requirements engineer*. Requirement engineers are responsible for eliciting, analysing, and managing the requirements, including the properties and relationships of the requirements. This task is assisted by the Requirements intelligence engine (WP2), which is in charge, among others, of the data and text mining of the requirements, and the Dependency engine (WP5), which is in charge of detection of requirements dependencies, requirements tracing, and resolution of conflicts to repair inconsistent requirements. More information about both engines can be found in deliverables D2.1 and D5.1, respectively.

The stakeholders' recommender engine operates in the context of other systems, namely *Requirement Management Systems* (RMS), as shown in Figure 5. In general, as thought in Openreq, the requirements are stored in the RMS. However, in some cases (e.g., for Siemens trial partner) the requirements will be automatically extracted out of a tender document and stored later in the RMS. In such cases, a document will be sent to the stakeholders' recommender engine through the RMS, and analysed by this engine to extract the requirements.

Figure 5 depicts the two key RE tasks for which the stakeholders' recommender engine offers assistance:

1. *Requirements specification*. During this task, the requirements engineer will receive recommendations about the screening of requirements, relevant requirements, improvement suggestions on the quality of requirements, and properties of requirements, giving place to the addition of requirements or the modification of the requirements and their properties.
2. *Stakeholder assignment*. New stakeholders for the requirement being edited will be recommended, giving place to new stakeholders' assignments. This assignment will be done not only to detect the stakeholders that could better contribute to a requirement, but also to increase stakeholders' engagement in RE tasks.

In addition, the stakeholders' recommender engine will have a *context-observer* component that will be used to monitor the behaviour of the stakeholders in the RMS. This context information will be used by the rest of the recommendations tasks provided inside the stakeholders' recommender engine.

3.3.2 Architecture overview of the stakeholders' recommender engine

Figure 6 presents an overview of the architecture for the stakeholders' recommender engine that will be part of OpenReq. However, this architecture will probably evolve as the implementation progresses.

As it can be seen in Figure 6, we follow a *microservice* architecture. We have organized the microservices in two layers: the *recommendation layer* and the *data layer*. In addition, we have considered the possible interactions of the services of the engine with services of other work packages. In the following, we discuss each layer in a separate subsection as well as the relation with other microservices.

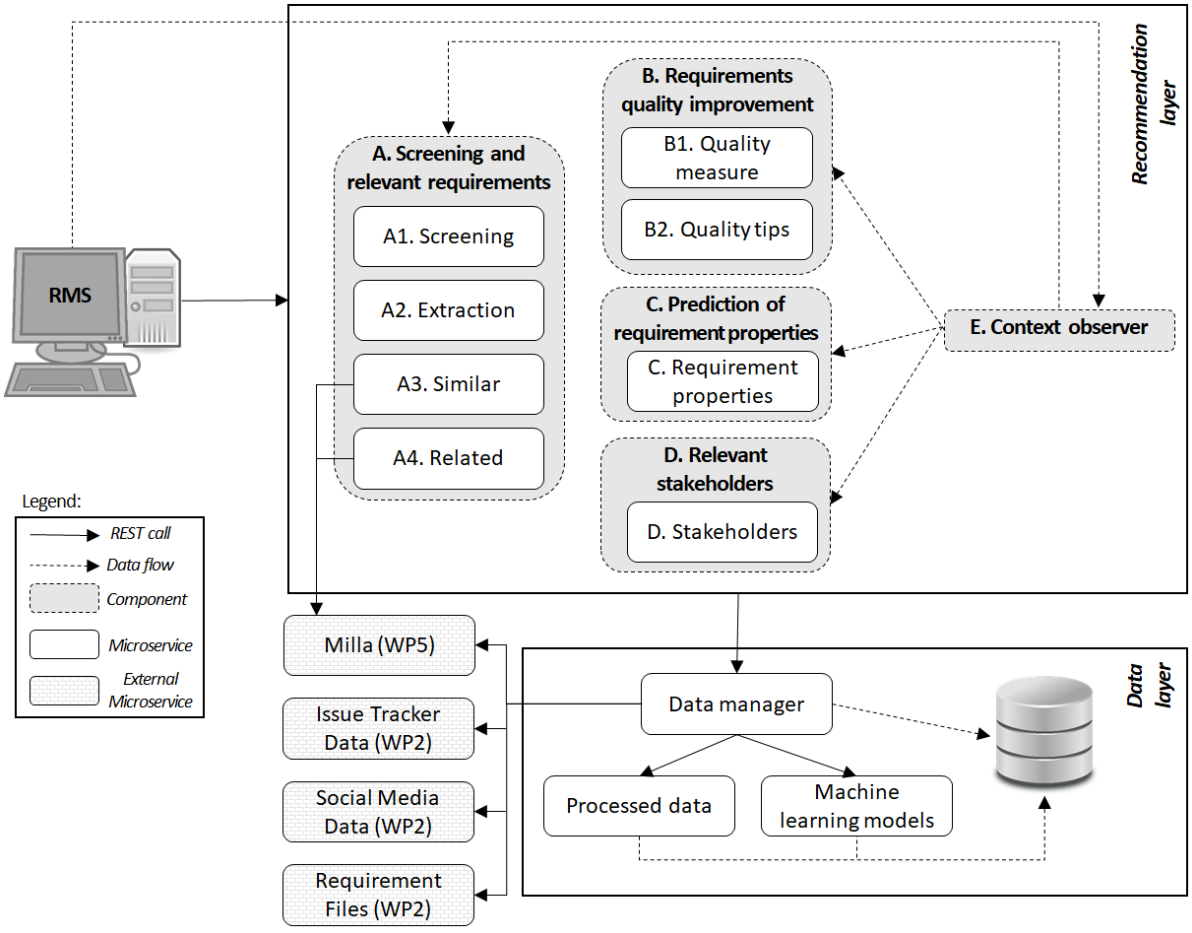

Figure 6. Architecture overview of the stakeholders' recommender engine

3.3.2.1 Recommendation layer

This layer is the interface to the frontend of OpenReq and includes all the microservices related to the different stakeholders' recommendations tasks that have been identified in subsection 3.1. We differentiate five components (one for each task in the DoA, i.e., T3.2, T3.3, T3.4, T3.5 and T3.6), and eight microservices. Specifically, the numeration of each microservice and component in this layer (Figure 6) follows the same numeration as in subsection 3.1, and,

therefore, each microservice and component implements the tasks introduced for its corresponding explanation in subsection 3.1.

The only thing to highlight in this layer is that microservices *A3. Similar* and *A4. Related* will communicate with the service of WP5, called *Milla*, which will provide, among other things, the existent dependencies among requirements. These dependencies, which also include the type *similar*, will help to provide the recommendations about similar and related requirements in microservices A3 and A4, respectively.

3.3.2.2 Data layer

All microservices in this layer (Figure 6) have the goal to persist the raw data necessary to run the microservices of the recommendation layer but also to store some intermediate results of the recommendation layer for easier and faster execution (e.g., avoid training a machine learning model every time it is needed). Figure 7 includes an initial overview of how the data will be organized — i.e., the *data model* for the stakeholders' recommender engine. As with the architecture, we do not discard that this model will evolve as the implementation progresses. The data model presented in Figure 7 is compliant with the first version of the OpenReq ontology presented in deliverable D5.1. Further details about the mapping of the data model in Figure 7 and the OpenReq ontology will be presented in further deliverables (specifically, in D5.4).

Part 1 of the model is formed by the raw data necessary to run the microservices of the recommendation layer. That includes information about the requirements projects (i.e., *Project*, *Requirement*, *Requirement Property*, and *Dependency*), from where this data came from (in case it was imported to the system) (i.e., *Issue*, *Social Media*, and *Requirement File*), and the stakeholders involvement (i.e.., *Stakeholder*, *Participation*[31], and *Requirement Rating*). The information about Requirement and Requirement Property will come from the *Milla* microservice of WP5, while the information Issue, Social Media, and Requirement File will come from the *Issues Tracker Data*, *Social Media Data* and *Requirement Files* microservices of WP2. *Part 2* of the model includes intermediate results of the recommendation layer. This part of the model, together with the services of this layer, are explained in the following paragraphs.

Data Manager microservice

This microservice is the entry point for the storage and retrieval of data, orchestrating all the actions needed in relation to these tasks (i.e., this microservice will decide what should be done and what other services to use from this layer). On one hand, it routes data from the recommendation layer to the appropriate microservice. On the other, it stores the data used by recommender systems and machine learning algorithms in the recommendation layer. From a data model perspective (see Figure 7), Part 1 of the model represents such abstracted data. The Data Manager helps decoupling the raw data from the models data. One challenge is that the Data Manager needs to be updated once a new type of machine learning model needs to be persisted.

[31] Participation refers to the intervention of a Stakeholder in a Project to rate the Requirement Properties of a Requirement.

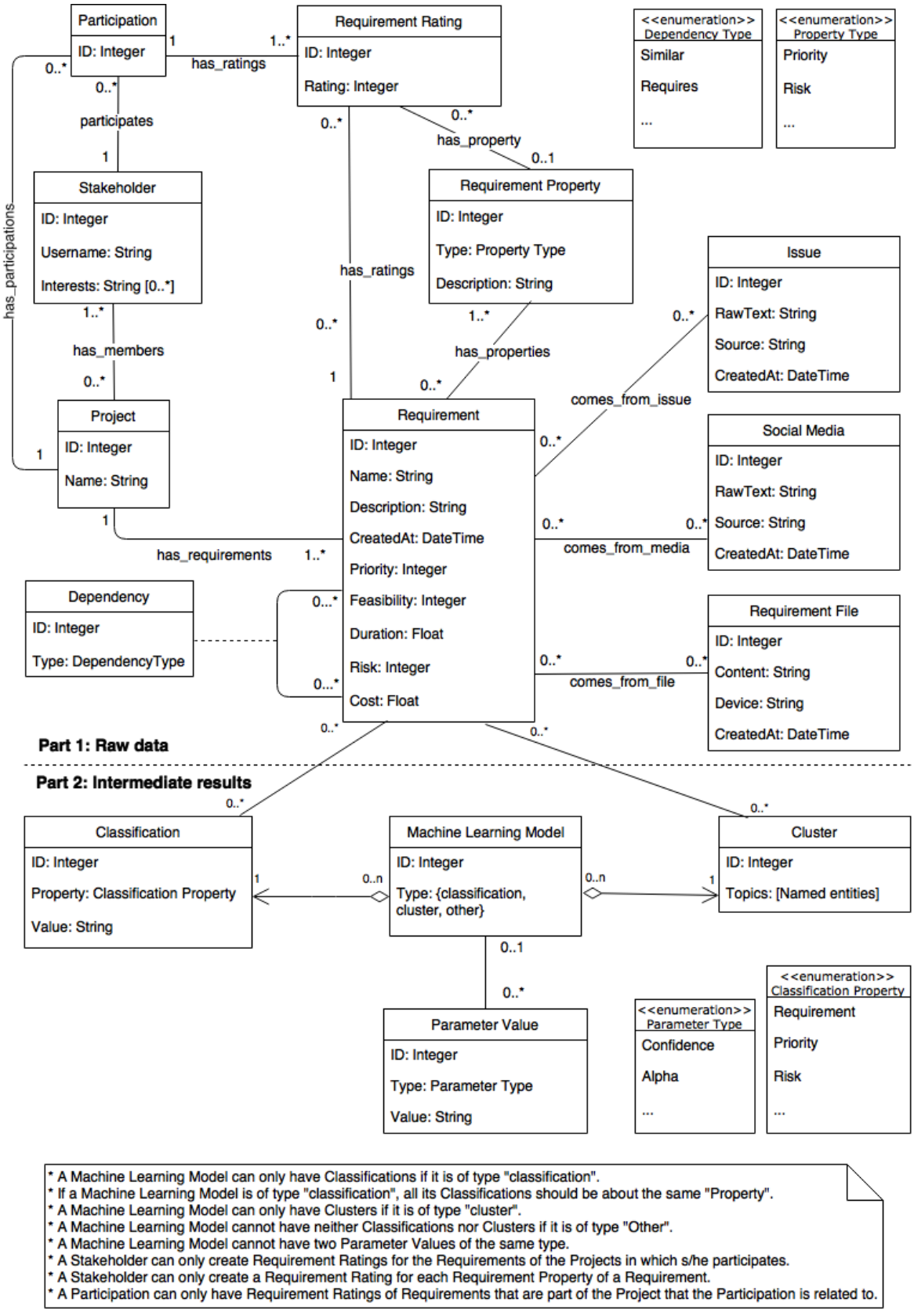

Figure 7. Data entity relationship model of the Data layer

Machine Learning Models microservice

This microservice avoids re-computing machine learning models needed by the recommendation layer. From a data perspective (see Figure 7), it stores the results of a machine learning algorithm in the appropriate entity — *Classification*, *Cluster*, or *Machine Learning Model*. The Classification entity stores data related to classification operations of the Requirement (e.g., requirements classification as actual requirements). Note that the model allows to classify the requirements according to different properties. Accordingly, Cluster will store information regarding automatically identified groups of requirements (e.g., requirements concerning the same topic). Finally, Machine Learning Model stores the results of a generic machine learning model that may operate on its own, or by using groups of clusters or classes of requirements.

Processed Data microservice

This microservice stores the machine learning features for training the machine learning models extracted from the requirements. This avoids, together with the Machine Learning Models microservice, re-computing the entire input when new data is received. Therefore, only the new data 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 7), this microservice stores and provides the *Parameter Value* (e.g., the confidence level) with which the Machine Learning Models have been computed. Such approach decouples the input data relative to the OpenReq requirements data from the model specific input. This microservice depends on the type of machine learning models used in the recommendations layer.

4 SUMMARY

In this deliverable we have presented the strategy devised to build the stakeholders' recommender engine to cover the expectations of the OpenReq project and its intended platform. Such results come from the development of Task 3.1 (Design stakeholders' recommendations approach).

First, we presented the state-of-the-art for recommender systems and requirements quality improvements. This state-of-the-art was carried out following two systematic mappings and includes the relevant results that could be of interest for developing the stakeholders' recommendation approach.

Afterwards, we summarized the recommendations tasks that the stakeholders' recommender engine is going to embrace, and gave details about the algorithms that will be used for each one of them. In a nutshell, we will implement a hybrid recommendation system which will combine classical recommendation techniques (such as content-based and knowledge-based techniques) with text mining, clustering and classification techniques.

In addition, we briefly introduced some evaluation mechanisms and metrics. The stakeholders' recommender engine will mostly be validated using offline evaluations with user studies, and metrics from the information retrieval field (e.g., precision and recall), trying to make our evaluations as reproducible as possible.

Finally, we proposed a microservice architecture for the engine, divided into two layers (i.e., recommendation layer and data storage layer), to enable highly decoupled software components, where each component will only focus on a small set of tasks. The components will expose their APIs, which can be used by third parties or by other components of the OpenReq platform.

5 REFERENCES

Adamopoulos P., and Tuzhilin, A. Recommendation opportunities: improving item prediction using weighted percentile methods in collaborative filtering systems.7th ACM conference on Recommender systems, ACM, 2013.

Adomavicius G., and Tuzhilin A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. on Knowl. and Data Eng., 17(6), pp. 734–749, 2005.

Adomavicius G., Tuzhilin A. Context-Aware Recommender Systems. Book chapter in Recommender Systems Handbook. Springer, 2011.

Aggarwal C.C., and Zhai C. An Introduction to Text Mining. In: Aggarwal C., Zhai C. (eds) Mining Text Data. Springer, 2012.

Alan S., Jain B.J., and Albayrak S. Analyzing weighting schemes in collaborative filtering: cold start, post cold start and power users. 27th Annual ACM Symposium on Applied Computing, ACM, 2012.

Antoniol G., Ayari K., Di Penta M., Khomh F., and Guéhéneuc Y.G. Is it a bug or an enhancement?: a text-based approach to classify change requests. Proceedings of the IBM Centre for Advanced Studies Conference on Collaborative Research, 2008.

Anvik J., Hiew L., and Murphy, G.C. Who should fix this bug? ACM/IEEE International Conference on Software Engineering, pp. 361–370, 2006.

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

Arora C., Sabetzadeh M., Briand L., and Zimmer F. Improving requirements glossary construction via clustering: approach and industrial case studies. 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014.

Baltrunas L., and Ricci F. Experimental evaluation of context-dependent collaborative filtering using item splitting. User Model User-Adap. Inter, 2014.

Basile P., Caputo A., and Semeraro G. Semantic vectors: an information retrieval scenario. Proceedings of the First Italian Information Retrieval Workshop (IIR), 2010.

Basili R., Bosco C., Delmonte R., Moschitti A. and Simi M. Harmonization and development of resources and tools for Italian natural language processing within the PARLI Project. Springer, 2015.

Basu C., Hirsh H., and Cohen W. Recommendation as classification: using social and content-based information in recommendation. 15th National Conference on Artificial Intelligence (AAAI'98), pp. 714–720, 1998.

Baeumer F.S., and Geierhos M. Running Out of Words: How Similar User Stories Can Help to Elaborate Individual Natural Language Requirement Descriptions. International Conference on Information and Software Technologies, pp. 549-558, 2016.

Beel J., Genzmehr M., Gipp B. A Comparative Analysis of Offline and Online Evaluations and Discussion of Research Paper Recommender System Evaluation. Workshop on

Reproducibility and Replication in Recommender Systems Evaluation (RepSys) at the ACM Recommender System Conference (RecSys), 2013.

Beel J., Breitinger C., Langer S., Lommatzsch A., and Gipp B. Towards reproducibility in recommender-systems research. User modeling and user-adapted interaction, 26.1, pp. 69-101, 2016.

Bell R.M., Bennett J., Koren Y., and Volinsky C. The million dollar programming prize. IEEE Spectrum, vol. 46, no. 5, pp. 28-33, 2009.

Bernardes D., Diaby M., Fournier R., Soulié F.F., and Viennet E. A Social Formalism and Survey for Recommender Systems. SIGKDD Explor. Newsl., 16(2), pp.20-37, 2015.

Berry D.M., and Kamsties E. Ambiguity in requirements specification. Perspectives on Software Requirements, pp. 7–44, 2004.

Berry D.M., Kamsties E., and Krieger M.M. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. Retrieved from http://www.osel.co.uk/papers/ambiguityhandbook.pdf. 2003.

Breese J.S., Heckerman D., and Kadie C.M. Empirical analysis of predictive algorithms for collaborative filtering. 14th Conference on Uncertainty in Artificial Intelligence, pp. 43–52, pp. 1998.

Bucchiarone A., Gnesi S., Lami G., Trentanni G., Fantechi A.: QuARS Express - A Tool Demonstration. 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08), 2008.

Burke R., Hammond K.J., and Young B.C. Knowledge-based navigation of complex information space. 13th National Conference on Artificial Intelligence (AAAI '96), pp. 462–468, 1996.

Burke R. Knowledge-Based Recommender Systems. Encyclopedia of Library and Information Systems, vol. 69, 2000.

Burke R. Hybrid recommender systems: survey and experiments. User modeling and user-adapted interaction, 12(4), pp. 331–370, 2002.

Campos P.G., Díez F., and Cantador, I. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. User Modeling and User Model User-Adap. Inter., 2014.

Castro-Herrera C., Cleland-Huang J., and Mobasher B. Enhancing Stakeholder Profiles to Improve Recommendations in Online Requirements Elicitation. 17th IEEE International Requirements Engineering Conference (RE'09), pp. 37–46, 2009.

Castro-Herrera C., and Cleland-Huang J. Utilizing Recommender Systems to Support Software Requirements Elicitation. 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE'10), ACM, 2010.

Chien J.T. Hierarchical Theme and Topic Modeling. IEEE Transactions on Neural Networks and Learning Systems. 27, pp. 565–578, 2016.

Cleland-Huang J., Dumitru H., Duan C., and Castro-Herrara C. Automated support for managing feature requests in open forums. Communications of the ACM - A View of Parallel Computing, 52(10), pp. 68–74, 2009.

Cubranic D., and Murphy G.C. Automatic bug triage using text categorization. 16th International Conference on Software Engineering & Knowledge Engineering, pp. 92–97, 2004.

Dalpiaz F., Snijders R., Brinkkemper S., Hosseini M., Shahri A., and Ali R. Engaging the crowd of stakeholders in requirements engineering via gamification. Gamification, pp. 123-135, Springer International Publishing, 2017.

de Gemmis M., Lops P., Musto C., Narducci F., Semeraro G. Semantics-Aware Content-Based Recommender Systems. Book Chapter in Recommender Systems Handbook, Springer, pp. 119-159, 2015.

del Sagrado J., and del Águila I.M. Stability prediction of the software requirements specification, Software Quality Journal, pp.1-21, 2017.

DeVries B., and Cheng B.H. Automatic detection of incomplete requirements via symbolic analysis. ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 385-395, 2016.

DeVries B., and Cheng, B.H. Automatic Detection of Incomplete Requirements Using Symbolic Analysis and Evolutionary Computation. International Symposium on Search Based Software Engineering, pp. 49-64, 2017.

Di Lucca G.A., Di Penta M., and Gradara S. An Approach to Classify Software Maintenance Requests. IEEE International Conference on Software Maintenance, pp. 93-102, 2002.

Dooms S. Dynamic generation of personalized hybrid recommender systems. 7th ACM conference on Recommender systems, ACM, 2013.

Dumitru H., Gibiec M., Hariri N., Cleland-Huang J., Mobasher B., and Castro-Herrara C. On-demand feature recommendations derived from mining public product descriptions. 33rd International Conference on Software Engineering, pp. 181-19, 2011.

Eckhardt J., Vogelsang A., Femmer H., and Mager P. Challenging incompleteness of performance requirements by sentence patterns. IEEE 24th International Requirements Engineering Conference (RE), pp. 46-55, 2016.

Ekstrand M.D., Riedl J.T., and Konstan J.A. Collaborative filtering recommender systems. Foundations and Trends® in Human–Computer Interaction, 4.2, pp.81-173, 2011.

Engelhardt, Paul E., and Fernanda Ferreira. "Processing coordination ambiguity." Language and speech 53.4, pp. 494-509., 2010.

Felfernig A., Friedrich G., Jannach D., and Zanker M. An integrated environment for the development of knowledge-based recommender applications. International Journal of Electronic Commerce 11, no. 2, pp. 11–34, 2006.

Felfernig A., Friedrich G., Schubert M., Mandl M., Mairitsch M., and Teppan, E. Plausible repairs for inconsistent requirements. 19th International Joint Conference on Artificial Intelligence (IJCAI'09), pp. 791–796, 2009.

Felfernig A., Schubert M., Reiterer S. Personalized diagnosis for over-constrained problems. International Joint Conference on Artificial Intelligence, pp. 1990–1996, 2013.

Felfernig A., Jeran M., Ninaus G., Reinfrank F., Reiterer S., and Stettinger M. Basic Approaches in Recommendation Systems. Chapter in Recommendation Systems in Software Engineering, Springer, Berlin, Heidelberg, 2014.

Femmer H., Fernández D.M., Juergens E., Klose M., Zimmer I., and Zimmer J. Rapid requirements checks with requirements smells: two case studies. 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014), pp. 10–19, 2014.

Fernández-Tobías I., Braunhofer M., Elahi M., Ricci F., and Cantador I. Alleviating the new user problem in collaborative filtering by exploiting personality information. User Modeling and User-Adapted Interaction, 26(2-3), pp.221-255, 2016.

Ferrari A., Spoletini P., and Gnesi S. Ambiguity Cues in Requirements Elicitation Interviews. IEEE 24th International Requirements Engineering Conference (RE), pp. 56–65, 2016.

Ferrari A., Donati B., and Gnesi S. Detecting Domain-Specific Ambiguities: An NLP Approach Based on Wikipedia Crawling and Word Embeddings. IEEE 25th International Requirements Engineering Conference Workshops (REW), pp. 393–399, 2017.

Fitzgerald C., Letier E., and Finkelstein A. Early failure prediction in feature request management systems. IEEE 19th International Requirements Engineering Conference (RE'11), pp. 229-238, 2011.

Gaeul J., Sunghun K., and Zimmermann T. Improving bug triage with bug tossing graphs. European Software Engineering Conference, pp. 111–120, 2009.

Gleich B., Creighton O., and Kof L. Ambiguity detection: Towards a tool explaining ambiguity sources. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 218–232, 2010.

Gunawardana A., Shani G. Evaluating Recommender Systems. In: Ricci F., Rokach L., Shapira B. (eds) Recommender Systems Handbook. Springer, Boston, MA. 2015.

Guy I. Social recommender systems. Recommender Systems Handbook, Springer US, pp.511-543, 2015.

Hofmann T. Latent Semantic Models for Collaborative Filtering. ACM Trans. Information Systems, 22(1), pp. 89-115, 2004.

Huertas C., Gómez-Ruelas M., Juárez-Ramirez R., and Plata H. A formal approach for measuring the lexical ambiguity degree in natural language requirement specification: Polysemes and Homonyms focused. International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE), vol. 1, pp. 115–118, 2011.

Hussein T., Linder T., Gaulke W., and Ziegler J. Hybreed: A software framework for developing context-aware hybrid recommender systems. User Modeling and User-Adapted Interaction, 24(1-2), pp. 121-174, 2014.

Jannach D., Zanker M., Felfernig A., and Friedrich G. Recommender systems: An introduction. Cambridge University Press, 2010.

Juergens E., Deissenboeck F., Feilkas M., Hummel B., Schaetz B., Wagner S., and Streit J. Can clone detection support quality assessments of requirements specifications? 32nd International Conference on Software Engineering, Vol. 2, pp. 79-88, 2010.

Kumar M., Ajmeri N., and Ghaisas S. Towards Knowledge Assisted Agile Requirements Evolution. 2nd International Workshop on Recommendation Systems for Software Engineering, pp. 16–20, 2010.

Lamkanfi A., Demeyer S., Giger E., and Goethals B. Predicting the severity of a reported bug. Mining Software Repositories, pp. 1–10, 2010.

Lamkanfi A., Demeyer S., Soetens Q.D., and Verdonck, T. Comparing mining algorithms for predicting the severity of a reported bug. Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 249–258, 2011.

Lathia N., Hailes S., Capra L., Amatriain X. Temporal diversity in recommender systems. 33rd International ACMSIGIR Conference on Research and Development in Information Retrieval (SIGIR 2010), pp. 210–217, 2010.

Lim S., and Finkelstein A. Stakerare: Using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Trans. on Softw. Eng., 38(3), pp. 707-735, 2012.

Linden G., Smith B., and York, J. Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Comput, 7(1), pp. 76–80, 2003.

Lombriser P. (2015). Engaging Stakeholders in Scenario-Based Requirements Engineering with Gamification. Master's thesis.

Lucassen G., Dalpiaz F., van der Werf J.M.E., amd Brinkkemper S. Forging high-quality user stories: towards a discipline for agile requirements. IEEE 23rd International Requirements Engineering Conference (RE), pp. 126-135, 2015.

Manning C.D., Surdeanu M., Bauer J., Finkel J., Bethard S.J., McClosky D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations, pp. 55–60, 2014.

Massey A.K., Rutledge R.L., Antón A.I., Hemmings J.D., and Swire P.P. A Strategy for Addressing Ambiguity in Regulatory Requirements. Retrieved from https://smartech.gatech.edu/bitstream/handle/1853/54573/ambiguity2.pdf?sequence=1&isAllowed=y. 2015.

McCarey F., Ó Cinnéide M., and Kushmerick, N. RASCAL: A recommender agent for agile reuse. Artif. Intell. Rev, 24(3–4), pp. 253–276, 2005.

Melville P., Mooney R.J., and Nagarajan R. Content-Boosted Collaborative Filtering for Improved Recommendations. 18th National Conference on Artificial Intelligence (AAAI), pp. 187–192, 2002.

Menzies T., AND Marcus, A. Automated severity assessment of software defect reports. IEEE International Conference on Software Maintenance, pp. 346–355, 2008.

Meuth R.J., Robinette P., and Wunsch D.C. Computational intelligence meets the NetFlix prize. EEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp. 686-691, 2008.

Mikolov T., Sutskever I., Chen K., Corrado G.S., and Dean J. Distributed Representations of Words and Phrases and their Compositionality. Retrieved from http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality. 2013.

Mooney R., and Roy L. Content-based book recommending using learning for text categorization. 5th ACM conference on Digital libraries (DL '00), ACM, pp. 195-204, 2000.

Musto C. Enhanced vector space models for content-based recommender systems. 4th ACM conference on Recommender systems, ACM, 2010.

Nagwani N.K., and Verma S. Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes. International Conference on ICT and Knowledge Engineering, pp. 113–117, 2012.

Ninaus, G., Felfernig, A., Stettinger, M., Reiterer, S., Leitner, G., Weninger, L., & Schanil, W. INTELLIREQ: intelligent techniques for software requirements engineering. In Proceedings of the Twenty-first European Conference on Artificial Intelligence (pp. 1161-1166). IOS Press, 2014.

Olga Baysal M. W., and Cohen G.R. A Bug You Like: A Framework for Automated Assignment of Bugs. 17th International Conference on Program Comprehension (ICPC '09), pp. 297–298, 2009.

Ott D., and Raschke A. Evaluating benefits of requirement categorization in natural language specifications for review improvements. Technical report, 2013.

Pennock D., Horvitz E., Lawrence S., and Giles C.L. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. 16th Conference on Uncertainty in Artificial Intelligence (UAI '00), pp. 473–480, 2000.

Petersen K., Feldt R., Mujtaba S., and Mattsson M. Systematic Mapping Studies in Software Engineering. 12th international conference on Evaluation and Assessment in Software Engineering (EASE), vol. 8, pp. 68-77, 2008.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. A design science research methodology for information systems research. Journal of management information systems, 24(3), 45-77, 2007.

Rashid A., Albert I., Cosley D., Lam S., McNee S., Konstan J., and Riedl J. Getting to know you: Learning new user preferences in recommender systems, 7th International Conference on Intelligent User Interfaces (IUI '02), ACM, pp.127–134, 2002.

Rehbein I., Ruppenhofer J., Sporleder C., and Pinkal M. Adding nominal spice to SALSA - frame-semantic annotation of German nouns and verbs. Conference on Natural Language Processing (KONVENS), 2012.

Ricci F., Rokach L., and Shapira B.. Recommender systems: Introduction and challenges. Recommender Systems Handbook, Springer US, pp. 1-34, 2015.

Romero-Mariona J., Ziv H., and Richardson D. J. SRRS: A Recommendation System for Security Requirements. International Workshop on Recommendation Systems for Software Engineering, 2004.

Sabriye A.O.J., and Zainon W.M.N.W. A framework for detecting ambiguity in software requirement specification. IEEE 8th International Conference on Information Technology, pp. 209–213, 2017.

Said A., Jain B.J., Albayrak S. Analyzing Weighting Schemes in Collaborative Filtering: Cold Start, Post Cold Start and Power Users. ACM Symposium on Applied Computing (SAC), 2012.

Swinney D.A. Lexical access during sentence comprehension: (Re)consideration of context effects. Journal of Verbal Learning and Verbal Behavior, 18(6), pp.645–659, 1979.

Tan P.N., Steinbach M, and Kumar V. Cluster Analysis: Basic Concepts and Algorithms. Book chapter in Introduction to Data Mining, Addison-Wesley, 2005a.

Tan P.N., Steinbach M, and Kumar V. Classification: Basic Concepts, Decision Trees, and Model Evaluation. Book chapter in Introduction to Data Mining, Addison-Wesley, 2005b.

Tso-Sutter K.H.L., Marinho L.B., and Schmidt-Thieme L. Tag-aware recommender systems by fusion of collaborative filtering algorithms. 2008 ACM Symposium on Applied Computing (SAC '08), ACM, pp. 1995–1999, 2008.

Unterkalmsteiner M., and Gorschek T. Requirements Quality Assurance in Industry: Why, What and How? International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), pp. 77-84, 2017.

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

Verstrepen K., and Goethals B. Top-N recommendation for shared accounts. P9th ACM Conference on Recommender Systems, ACM, 2015.

Weiß C., Premraj R., Zimmermann T., and Zeller, A. How Long Will It Take to Fix This Bug?. 4th Working Conference on Mining Software Repositories (MSR'07), 2007.

Wilmink M., and Bockisch C. On the ability of lightweight checks to detect ambiguity in requirements documentation. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 10153 LNCS, pp. 327–343, 2017.

Wilson W.M., Rosenberg L. H., and Hyatt L.E. Automated analysis of requirement specifications. 19th international conference on Software engineering (ICSE), pp. 161–171, 1997.

Winkler J., Vogelsang A. Automatic Classification of Requirements Based on Convolutional Neural Networks. IEEE 24th International Requirements Engineering Conference Workshops (REW'16), 2016.

Yang H., Willis A., De Roeck A., Nuseibeh B., and De Roeck A. Automatic detection of nocuous coordination ambiguities in natural language requirements. IEEE/ACM international conference on Automated software engineering (ASE), pp. 53–62, 2010.

Zhang Y., and Pennacchiotti M. Recommending branded products from social media. 7th ACM conference on Recommender systems, ACM, 2013.