An Overview of Recommender Systems in Requirements Engineering

Alexander Felfernig[1], Gerald Ninaus[1], Harald Grabner[1], Florian Reinfrank[1], Leopold Weninger[2], Dennis Pagano[3], and Walid Maalej[3]

Abstract Requirements Engineering (RE) is considered as one of the most critical phases in software development. Poorly implemented RE processes are still one of the major risks for project failure. As a consequence, we can observe an increasing demand for intelligent software components that support stakeholders in the completion of RE tasks. The aim of this chapter is to give an overview of the research dedicated to the application of recommendation technologies in RE. On the basis of an analysis of related work we exemplify the application of recommendation technologies in different scenarios. In this context we focus on the approaches of collaborative filtering, content-based filtering & clustering, knowledge-based recommendation, and group-based recommendation & social network analysis.

1 Introduction

Requirements Engineering (RE) can be defined as *the branch of systems engineering concerned with the desired properties and constraints of software-intensive systems, the goals to be achieved in the softwares environment, and assumptions about the environment* [10]. Core activities of an RE process are elicitation & definition, validation, negotiation, and release planning [44]. RE is considered one of the most critical phases of a software development process and poorly implemented RE is one of the major reasons for the failure of a project [24].

Due to the increasing size and complexity of software systems, we can observe a growing demand for intelligent methods, techniques, and software systems that can

[1]Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b, A-8010, Graz, Austria, e-mail: {alexander.felfernig, gerald.ninaus, harald.grabner, florian.reinfrank}@ist.tugraz.at
[2]wsop, Muellnergasse 4, A-1090, Vienna, Austria, e-mail: {LWeninger}@wsop.at
[3]Applied Software Engineering, Technische Universitaet Muenchen, Boltzmannstrasse 3, D-85748 Garching, Germany, e-mail: {dennis.pagano, walid.maalej}@in.tum.de

help to improve the overall quality of RE processes [3, 17]. In this chapter we focus on the aspect of how recommendation technologies [38] can be applied to support stakeholders in different phases of RE processes. Recommendation technologies are intensively applied for the purpose of recommending products and services, for example, movies, books, digital cameras, or financial services. The application of these technologies in RE scenarios is an upcoming research area [3]. A recommender system can be defined as *any system that guides a user in a personalized way to interesting or useful objects in a large space of possible options or that produces such objects as output* [5, 38].

The remainder of this chapter is organized as follows. In Section 2 we provide an overview of existing research on the application of recommendation technologies in RE. In Section 3 we discuss different application scenarios for recommendation technologies with a focus on collaborative filtering [28], content-based filtering [37] & clustering [46], knowledge-based recommendation [5, 14], and group-based recommendation [34] & social network analysis [22]. In Section 4 we discuss relevant issues for future research. The chapter is concluded with Section 5.

2 Research on Recommender Systems in Requirements Engineering

In this section we discuss existing research dedicated to the application of recommendation technologies in Requirements Engineering (RE). Our discussion of related research is organized along the typical activities in a RE process – we take into account the activities of *requirements elicitation & definition*, *quality assurance*, and *negotiation & release planning*. In order to provide further technical insights into the recommendation approaches discussed in this chapter, we provide examples for their application in Section 3.

2.1 Requirements Elicitation & Definition

Requirements elicitation & definition focuses on the collection of requirements from different stakeholders. Typical resulting artifacts are, for example, textual requirement descriptions, scenario descriptions, use cases, and sketches of prototypical user interfaces. The following recommendation approaches support activities related to requirements elicitation & definition.

Recommending Stakeholders. This is an important task in the early phases of an RE process since, for example, a low degree of user involvement in most cases leads to project failure [31]. The major goal of stakeholder identification is to identify a set of persons who are capable of providing a complete and accurate description of the software requirements. Identifying a set of authorized, collaborative, responsible, committed, and knowledgeable stakeholders is a very important and challenging

task [3, 31]. A common mistake is that the wrong representatives of groups are integrated into a project or that stakeholders relevant for the project are simply omitted.

StakeNet [31] is an approach to stakeholder identification which is based on the concepts of social network analysis [22, 33]. In *StakeNet*, an initial set of stakeholders and recommendation information provided by these stakeholders is applied for the construction of a social network (SN) where the included nodes represent the stakeholders and connections between nodes represent recommendations articulated by the stakeholders, i.e., if stakeholder s_i recommends stakeholders s_j with a certain rating then this information is included in the corresponding social network. This process of stakeholder recommendation is repeated in order to exploit a kind of snowball effect. On the basis of the constructed social network, different SN analysis algorithms are exploited for stakeholder prioritization. An example of such an analysis approach is *betweenness centrality* which measures for a specific stakeholder s_i the number of shortest paths between other stakeholders in which s_j is contained. A high value of this measure indicates a person's capability of acting as a broker between different groups of stakeholders.

Especially in large-scale and distributed software projects it is infeasible to organize personal meetings on a regular basis. In such scenarios requirements are often defined in Wiki-based forums which are very receptive to the problems of information overload, redundancy, incompleteness of information, and diverging opinions of different stakeholders. In their approach to improve the stakeholder support in *ultra-large-scale software systems* development (ULS software systems), Cleland-Huang et al. [3, 7] introduce concepts for clustering user requirements and in the following to assign (recommend) stakeholders to clusters on the basis of content-based filtering [37]. One major motivation for such an assignment of stakeholders to requirement clusters is to achieve a representative coverage, i.e., each requirement should be discussed and evaluated by a sufficient number of stakeholders.

Recommending Requirements. A systematic reuse of already existing software requirements has the potential of significantly reducing the overall costs of a software project. A recommendation-based approach to requirements reuse is presented by Dumitru et al. [12]. The basic idea is to analyze requirements which are accessible in software project repositories and to apply clustering algorithms for the intelligent grouping of such requirements. The identified requirement groups can be analyzed in future software projects for the purpose of reuse and also for the purpose of completeness checking (are all relevant requirements contained in the current requirements model). The proposed recommendation approach is content-based filtering, where a vector of keywords (derived from the description of the new software project) is matched with the keywords extracted from requirements artifacts from the repository of already completed software projects.

Lim and Finkelstein [30] introduce the *StakeRare* approach which supports the identification and reuse of requirements. *StakeRare* [30] is based on the afore mentioned *StakeNet* approach [31]. In *StakeRare* stakeholders are rating initial sets of requirements. Additional (new) requirements currently not contained in the list are then recommended using the concepts of collaborative filtering [28]. On the basis

of the rating information (weighted with the stakeholders weight (influence) in the current project) requirements are prioritized.

Similar to their approach of recommending (assigning) stakeholders to requirement clusters (topics), the requirements engineering environment discussed in Cleland-Huang et al. [3, 7] also supports the recommendation of requirements to stakeholders, for example, based on the concepts of collaborative filtering. A major motivation for the application of collaborative filtering in this scenario was to achieve serendipity effects which help to increase requirements quality (stakeholders receiving recommendations regarding requirements they are interested in, have a higher proability of analyzing these requirements). Another motivation for the application of collaborative filtering is to improve requirement model understanding since it generates personalized navigation paths for stakeholders.

2.2 Quality Assurance

A set of requirements has to be evaluated regarding properties such as consistency (requirements are not contradictory), completeness (all relevant requirements should be part of the requirements model), feasibility (technical feasibility as well as economic feasibility), understandability (does the description fulfill the quality standards), and reusability (are the requirements reusable in future projects). Currently, recommenders are applied to support the following quality assurance scenarios.

Managing Feature Requests. The major goal of feature request management is to support the effective management of large sets of software features. Unstructured request management can lead to suboptimal communication between stakeholders and to the selection of in the worst case irrelevant features [3]. An approach to support effective feature management has been introduced by Cleland-Huang et al. [8] where clusters of similar requirements are exploited for the identification of redundancies and the prioritization of feature requests. Fitzgerald et al. [21] introduce an approach to feature request management which is based on the idea of predicting software failures (e.g., anbandoned implementation of a feature) by analyzing the communication threads in feature management systems. Their approach to failure identification is based on the idea of applying different machine learning algorithms for the construction of a prediction model for failures. The prediction model is based on an analysis of existing feature requests and their related positive or negative outcomes. Prediction models are derived on the basis of parameters that are assumed to be important for specifying the quality of a feature request, for example, *involvement of the right stakeholders* or *sufficient engagement of stakeholders* in terms of contributing to a feature-related discussion thread.

Consistency Management. Inconsistencies between requirements are resulting from factors such as not enough time for consistency checking, different perceptions and goals, or different granularity of knowledge. Especially for informally defined requirements the complete automation of consistency management is unrealistic [25] but semi-automated tools help to keep the efforts acceptable. Assum-

ing the existence of a formal description of the requirements model (on the basis of a constraint satisfaction problem [45]) and the stakeholder preferences (priorities) regarding the defined set of requirements, Felfernig et al. [19] introduce an approach to the automated diagnosis of inconsistent requirement models and inconsistent stakeholder preferences. In this context, a diagnosis is interpreted as a minimal set of stakeholder preferences (or requirements) that have to be adapted or deleted in order to restore consistency. A detailed introduction to the concepts of model-based diagnosis can be found, for example, in the work of Reiter [40].

Dependency Detection. Due to the informal nature of requirement models, the analysis regarding properties such as model consistency and completeness are challenging. Relationships between requirements are typically expressed in terms of dependencies (e.g., requirement *A* requires requirement *B* or requirement *A* is incompatible with requirement *B*) which are defined by stakeholders [9]. Recommender systems allow the provision of additional information which proactively supports stakeholders in the identification of dependencies. Dependency detection between requirements can be based, for example, on clustering techniques where requirements are grouped into clusters of similar topics (see, for example, Cleland-Huang et al. [8]). The basic underlying assumption is that requirements which are assigned to the same cluster are depending on each other. Although helpful, this approach does not result in a complete specification of the type of dependency but serves as a basis for a further analysis by stakeholders. Some preliminary work regarding requirements and inconsistency discovery and classification in *Open Source Software Development (OSSD)* which is based on the methods of Natural Language Processing (NLP) is presented by Fantechi and Spinicci [13].

2.3 Requirements Negotiation & Release Planning

Requirements negotiation is the process of identifying conflicts between stakeholder preferences and to facilitate efficient stakeholder decision making regarding priorities and acceptance (this process is also denoted as *requirements triage*). The major goal of release planning is the development of a schedule which specifies in which development (release) period which requirement should be implemented.

Requirements Triage. Restrictions regarding the available resources (e.g., budget and employees) and defined deadlines for the completion of a software system in many cases require decisions regarding the set of requirements which should implemented. Requirements have to be prioritized in order to take aside unimportant requirements and to support project managers in conflict resolution and making tradeoffs. Prioritization of requirements is an often complex and iterative communication and decision process [2] which has to take into account different soft factors such as company policies, personal preferences, and social relationships between stakeholders. *Requirements triage* is a term which stems from medical decision making [10]. In disaster scenarios victims are categorized into three different types: those who will die (independent of the medication), those who will survive (independent

of the medication), and those whose survival depends on the given medication. Requirements prioritization has to deal with a similar task: identify the requirements which must not be included in the next release, requirements that are optional for the next release, and the requirements that must be included in the next release.

The lack of efficient triage processes in large software projects with hundreds of stakeholders and thousands of (sometimes conflicting) requirements lead to the development of intelligent technologies supporting the semi-automated requirements prioritization. The approach presented by Duan et al. [11] focuses on the generation of clusters which are derived from different clustering criteria. The weight of different clustering criteria is specified by stakeholders and an initial prioritization is generated on the basis of a utility function which is based on the number of clusters a requirement is included in and the weights of the corresponding clusters.

Further Decision Support Approaches. Traditional models of human decision-making are based on the assumption that humans are taking decisions on the basis of rational thinking [36]. That is, a human would take the optimal decision following a formal evaluation process. One major assumption is that preferences remain consistent and unchangeable. In contradiction to these models, research has clearly pointed out the fact that preference stability in decision processes does not exist, and can also be easily manipulated [4]. A customer who wants to purchase a digital camera could first define a strict upper limit for the price. But due to additional technical information about the camera the customer could change her mind and significantly increase the upper limit of the price. This typical example of preference reversal [29] indicates the non-existence of stable preferences. Instead, the model of preference construction [4] should be used, in which decision making processes are more characterized by a process of iterative refinement and adaptation of the current preferences in the face of new alternatives and as well in the face of opinions of other users that are visible to the decision maker.

The idea of applying group decision making algorithms in Requirements Engineering is to exploit basic decision heuristics [35] such as *majority voting* (the decision is taken conform to the majority of the votes of the engaged stakeholders) or the *fairness heuristic* which guarantees that none of the stakeholders will be disadvantaged in the group decision process. Group decision heuristics already play an important role in application scenarios outside software engineering [35]. Felfernig et al. [20] applied group decision heuristics in the context of RE scenarios. They introduce the *IntelliReq* environment which can be used for supporting group decision process in distributed settings (e.g., open source platforms or large and distributed software projects). The authors present the results of an empirical study which show that group recommendation technologies can help to improve the perceived quality of decision support. A further insight was that stakeholders should not be confronted with the preferences of other group members at the beginning of prioritization – the reason is that knowledge about preferences automatically triggers insufficient information exchange between group members.

Recommendation of Release Plans. Ruhe et al. [42] introduce an approach to release planning that is based on the concept of linear programming [43]. The basic idea is to define a linear program that should calculate a sequence of assignments

Table 1 Overview of recommendation approaches for Requirements Engineering.

Scenario	Recommendation Approach	
recommending stakeholders	social network analysis	Lim et al. [31]
	content-based filtering	Castro et al. [7, 3]
recommending requirements	content-based filtering	Dumitru et al. [12]
	social network analysis	Lim and Finkelstein [30]
	collaborative filtering	Castro et al. [7, 3]
managing feature requests	clustering	Cleland-Huang et al. [8]
	machine learning	Fitzgerald et al. [21]
consistency management	knowledge-based	Felfernig et al. [19]
dependency detection	clustering	Cleland-Huang et al. [8]
requirements triage	clustering utility theory	Duan et al. [11]
negotiation & planning	group recommendation utility theory	Felfernig et al. [20]

of features to a corresponding release taking into account the dependencies between the different features. Ruhe et al. [41] show how to apply AHP (Analytical Hierarchy Process) for determining a set of preferred requirements. This work is an important contribution to improve the quality of requirements selection but depends on the assumptions that stakeholders know their preferences and that preferences remain stable. Felfernig et al. [16, 19] extend the work of Ruhe et al. [41] by introducing automated diagnosis and repair mechanisms which effectively help to figure out minimal sets of acceptable changes in situations where release plan preferences of stakeholders become inconsistent.

Table 1 provides an overview of the discussed application scenarios of existing recommendation approaches for requirements engineering.

3 Recommendation Algorithms for Requirements Engineering

In order to show how recommendation technologies can be exploited in the RE context, we will now introduce basic application scenarios. These scenarios should help to develop an understanding of potential applications of recommendation technologies and show how different recommendation approaches have to be tailored in order to be applicable. Note that we interpret recommendation technologies as key supportive technologies; we do not claim that information gaps in general can be closed by the application of recommendation technologies. On the one hand information does not substitute communication, i.e., effective RE processes still heavily rely on personal stakeholder interaction. Furthermore, the quality of recommendations depends on the quality of information provided by stakeholders, i.e., the successful application of recommendation technologies is only possible on the basis of motivated and proactive stakeholders. Finally, successful RE strongly depends on process quality, which can not be achieved and guaranteed only by the application of recommendation technologies. In the following we discuss basic RE application scenarios for the major types of recommendation technologies which are *content-*

Table 2 Example of a content-based filtering recommendation problem.

requirement	category	planned release	efforts (mds)	description
r_1	database	1	150	store component configuration in DB
r_2	user interface	2	60	user interface with online help available
r_3	database	1	300	separate tier for DB independence
r_4	user interface	1	30	user interface with corporate identity

based filtering (CBF) [37] & *clustering* [46], *collaborative filtering* (CF) [28], *group recommendation* (GR) [26] & *social network analysis* [22], and *knowledge-based recommendation* (KBR) [5, 14].

Content-based Filtering. Content-based filtering (CBF) [37] exploits the similarities between the preferences of the current user and descriptions of items the user did not notice up to now. User preferences can be, for example, represented by frequent keywords extracted from artifacts previously processed by the user. Another alternative are predefined categories assigned to items as meta-information. Typical recommendations derived by CBF recommenders are of the form *item C is recommended since you were also interested in item A* (which is similar to item *C*).

When *defining requirements*, a recommender can support stakeholders, for example, by indicating similar requirements or point out requirements already defined in previous projects. Let us assume, the active stakeholder (s_1) has already investigated the requirement r_1 which has the assigned category *database* (see Table 2). Now, CBF would recommend requirement r_3 if r_3 has not been investigated up to now by the active stakeholder. If no such categorization of requirements is available, the detailed textual description of requirements can as well be used: keywords have to be extracted [39] and the determination of similar requirements can then be based on the similarity of the extracted keywords – a simple corresponding similarity metric is shown in Formula 1. For example, sim(r_1, r_3) = 0.17, if we assume *keywords*(r_1) = {*store, component, configuration, DB*} and *keywords*(r_3) = {*tier, DB, independence*}.

$$sim(s,r) = \frac{|keywords(s) \cap keywords(r)|}{|keywords(s) \cup keywords(r)|} \tag{1}$$

k-Means Clustering. A basic method for determining clusters is *k-means clustering* [46] where *k* specifies the number of clusters being sought. In the initial iteration two requirements can be chosen as *cluster centers* and the other requirements are assigned to their closest cluster. Different distance metrics can be applied [46] – for the purposes of our example we apply the similarity between keywords (see Table 3) extracted from the textual description of our example requirements ($\{r_1, r_2, r_3, r_4\}$ in Table 2). Thereafter the centroid (mean) per cluster is determined for each cluster and an assignment of requirements to clusters takes place again. For our example we assume that after one step the two clusters c_1:$\{r_1, r_3\}$ and c_2:$\{r_2, r_4\}$ have been identified where $sim(r_1, r_3) = 0.17$ and $sim(r_2, r_4) = 0.5$.

Collaborative Filtering. Collaborative filtering (CF) [28] is perhaps the most widespread recommendation approach where information about the rating behavior

Table 3 Keywords extracted from the textual requirement descriptions in Table 2.

requirement	extracted keywords
r_1	store component configuration DB
r_2	user interface help
r_3	tier DB independence
r_4	user interface corporate

of *nearest neighbors* (i.e., users with similar ratings compared to the current user) is exploited for predicting the current user's ratings for items not known to her/him yet. Typical recommendations derived by CF recommenders are of the form *users who were interested in item A were also interested in item C*.

Table 4 Example of a collaborative recommendation problem. A table entry *ij* with value 1 (0) denotes that fact that stakeholder s_i has (has not) inspected the requirement r_j.

	r_1	r_2	r_3	r_4
s_1	1	0	1	0
s_2	1	0	1	1
s_3	1	1	0	1

When stakeholders try to *understand a given set of requirements* (e.g., new stakeholders in the project), recommender systems can provide support in terms of showing related artifacts or showing those artifacts stakeholders have investigated when working on the current or a similar requirement. In the setting of Table 4 the requirements $\{r_1, r_2, r_3, r_4\}$ have already partially been investigated by the stakeholders $\{s_1, s_2, s_3\}$, for example, stakeholder s_1 has already investigated the requirements r_1 and r_3. The main idea of collaborative filtering (CF) is to exploit user ratings (in our context the rating = 1 if a stakeholder has already investigated a certain requirement and the rating = 0 if the stakeholder did not investigate the requirement up to now) in order to identify additional requirements the stakeholder may be interested in. *User-based CF* is a basic variant which is often used in industrial contexts [28]. User-based CF tries to identify the k-nearest neighbors of the active stakeholder (stakeholders interested in a similar set of requirements) and calculates a prediction for the active stakeholder rating for an item the stakeholder has not investigated up to now. Such a rating can be defined, for example, as the weighted majority of the k-nearest neighbors. In our example, stakeholder s_2 can be identified as the nearest neighbor (if we set k=1) since s_2 has investigated all the requirements investigated by stakeholder s_1. Vice versa, stakeholder s_1 did not investigate the requirement r_4 up to now – in this context, our collaborative filtering algorithm would recommend requirement r_4 to stakeholder s_1 since the nearest neighbor of s_1 has already investigated r_4.

Group Recommendation. Group recommendation (GR) [26] can support groups in their decision process by taking into account the fact that individual decisions depend on various factors, such as own evaluation of a solution alternative, beliefs about the opinions of group members, and information about the individual motivation (e.g., egocentric or cooperative motivation [26]). GR includes algorithms and heuristics that can be exploited for identifying solution alternatives that are (with a high probability) accepted by all or at least the majority of group members, i.e., the

major goal of GR technologies is to support/achieve consensus among group members. Typical recommendations derived by group recommenders are of the form *this recommendation tries to take into account the preferences of all group members*.

Requirements evaluation & negotiation have a clear need of group decision support: a group of stakeholders has to decide about the quality of individual requirements and in the following to figure out which requirements should be taken into account. Let us assume that the requirement *r* has to be evaluated by the stakeholders $\{s_1, s_2, s_3, s_4\}$ – the individual evaluations of *r* are depicted in Table 5.

Table 5 Example of a decision problem: deciding about the group evaluation of requirement *r*.

requirement: *r*	s_1	s_2	s_3	s_4
quality	medium	medium	medium	high
effort (mds)	10	7	14	8
decision	accept	revision	accept	accept

In this context, group recommendation concepts can be applied which propose alternatives to be further evaluated by the group. Different strategies for determining such a group recommendation are possible [34], for example, the *least-misery strategy* would propose evaluations that are stable in the sense that none of the evaluation dimensions has been over-estimated (or under-estimated, for example, in the case of *mds* – man days). Applying this strategy in our context would mean to propose the evaluation (*quality* = *medium*, *effort* = *14*, *decision* = *revision*) as first alternative for the overall group decision. On the basis of this and further proposals each individual stakeholder enters the next review round with the goal to achieve (if possible) a consensus regarding the evaluation. A detailed discussion of further strategies for determining group recommendations can be found in Masthoff [34].

Social Network Analysis. With the concepts of Social Network Analysis different properties of a network of stakeholders engaged in a RE process can be identified. In order to sketch the analysis of *betweenness centrality* of stakeholders, we introduce the communication patterns between the stakeholders $\{s_1, s_2, s_3, s_4, s_5\}$ in Table 6. For simplicity, we assume that each discussion thread (related to one requirement) includes at most four comments and stakeholder s_i is connected to stakeholder s_j in a social network (see Figure 1) if both are in at least one common discussion thread. *Betweenness centrality* measures for each stakeholder s_i the number of shortest paths between pairs of other stakeholders s_j ($s_i \neq s_j$) in which s_i is included. Table 7 depicts the results of the *betweenness centrality* evaluation in our working example; the stakeholders s_1 and s_3 have a centrality measure of 3.0 whereas the other ones have a measure of 0.0.

Table 6 Communication patterns (e.g., in a discussion forum) between stakeholders $\{s_1, s_2, s_3, s_4, s_5\}$ regarding requirements $\{r_1, r_2, r_3, r_4\}$.

requirement: *r*	comment 1	comment 2	comment 3	comment 4
r_1	s_1	s_2	s_1	s_2
r_2	s_3	s_4	s_1	s_3
r_3	s_3	s_5	s_3	s_5
r_4	s_3	s_1	s_3	s_1

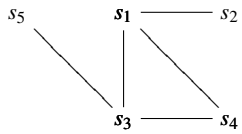

Fig. 1 Example Social Network (SN) derived from communication patterns of Table 6.

Table 7 *Betweenness Centrality* values for stakeholders $\{s_1, s_2, s_3, s_4, s_5\}$.

stakeholder	shortest paths between s_j	betweenness centrality
s_1	$s_2 - s_3$, $s_2 - s_4$, $s_2 - s_5$	3.0
s_2	–	0.0
s_3	$s_1 - s_5$, $s_2 - s_5$, $s_4 - s_5$	3.0
s_4	–	0.0
s_5	–	0.0

Knowledge-based Recommendation. Knowledge-based recommendation (KBR) [5, 14] exploits deep knowledge about the offered item assortment, knowledge about user preferences, and knowledge about which items should be recommended in which context. The explicit form of knowledge representation allows the generation of deep explanations as to why a certain item has been recommended or why no solution exists in a certain recommendation context [16]. Typical recommendations derived by KBR are of the form *you specified the item properties I*={*x,y,z*} *therefore we recommend* C which supports all the properties of *I*.

Recommendation technologies can support consistency management as well intelligent explanations in situations where no release plan can be identified due to contradicting stakeholder preferences [16, 19]. Table 8 depicts a set of requirements R={r_1,r_2,r_3,r_4} and a set of stakeholders S={s_1,s_2,s_3}. For each requirement $r_i \in$ R each stakeholder specifies his/her preferences which can be 1 (*include*) and 0 (*exclude*), for example, c_{12}=1 denotes the fact that stakeholder s_1 wants to include requirement r_2 in the next software release. The set of stakeholder preferences is denoted as C=$\cup c_{ij}$. Inclusion and exclusion are example constraints (preferences). Further types of constraints are possible (see, e.g., the RE ontology proposed by Lohmann et al. [32]) but not used in this example. For the preferences shown in Table 4 there does not exist a solution, i.e, the stakeholder preferences are inconsistent.

Table 8 Example of inconsistent stakeholder preferences: each table entry represents a constraint c_{ij}, where c_{ij} = 1 (0) denotes the fact that stakeholder *i* wants to include (exclude) requirement *j*.

	s_1	s_2	s_3
r_1	1	1	1
r_2	1	0	1
r_3	0	0	1
r_4	1	1	1

The first step to resolve this inconsistency is to figure out combinations of constraints (preferences) that are responsible for the inconsistency, for example, the

Table 9 Example importance values for the stakeholder preferences shown in Table 8.

	s_1	s_2	s_3
r_1	imp(c_{11})=0.5	imp(c_{21})=0.3	imp(c_{31})=0.4
r_2	imp(c_{12})=0.2	imp(c_{22})=0.3	imp(c_{32})=0.2
r_3	imp(c_{13})=0.2	imp(c_{23})=0.2	imp(c_{33})=0.2
r_4	imp(c_{14})=0.1	imp(c_{24})=0.2	imp(c_{34})=0.2

Table 10 Utility values of repair actions $\{repc_1, repc_2, repc_3, repc_4\}$.

$repc_k \in REP_c$	utility($repc_k$)
$repc_1$	2
$repc_2$	1.42
$repc_3$	1.66
$repc_4$	1.25

stakeholder preference c_{12} is inconsistent with the preference c_{22}. The complete set of such (minimal [27]) inconsistencies is CON = $\{con_1{:}\{c_{12}, c_{22}\}, con_2{:}\{c_{22}, c_{32}\}, con_3{:}\{c_{13}, c_{33}\}, con_4{:}\{c_{23}, c_{33}\}\}$. Such sets can be determined using the algorithm presented by Junker [27]. We can now determine all possible repairs for the given set C of stakeholder preferences by simple deleting at least one element from each subset of CON (see [40]). The possible repair constraint sets rep_k for CON are elements of REP = $\{rep_1{:}\{c_{22}, c_{33}\}, rep_2{:}\{c_{22}, c_{13}, c_{23}\}, rep_3{:}\{c_{12}, c_{32}, c_{33}\}, rep_4{:}\{c_{12}, c_{32}, c_{13}, c_{23}\}\}$ where a repair constraint set rep_k is defined as a *minimal set of stakeholder preferences* (see [18]) that have to be changed in order to make the stakeholder preferences consistent.

For the given set REP we can identify the following set of concrete repair actions REP_c= $\{repc_1{:}\{c_{22}{=}1, c_{33}{=}0\}, repc_2{:}\{c_{22}{=}1, c_{13}{=}1, c_{23}{=}1\}, repc_3{:}\{c_{12}{=}0, c_{32}{=}0, c_{33}{=}0\}, repc_4{:}\{c_{12}{=}0, c_{32}{=}0, c_{13}{=}1, c_{23}{=}1\}$. REP_c can now be considered as a set of alternative and minimal repairs for the original set of stakeholder preferences such that consistency between the preferences can be restored.

4 Issues for Future Research

Based on our analysis of existing research on the application of recommendation technologies in Requirements Engineering, we now focus on a discussion of relevant *issues for future research*.

Decision Support & Preference Construction. Existing requirements engineering approaches rely on the assumption of stable stakeholder preferences (e.g., in the context of requirements negotiation). The assumption of stable preferences is not applicable for Requirements Engineering scenarios, in fact, related decision making follows an incremental preference construction process [15, 20]. In order to better integrate recommendation technologies into requirements engineering processes, we are in the need of deep knowledge about human decision strategies. Such a knowledge will help us to improve the decision support quality. The integration

of human decision strategies into recommendation is a new and challenging field of research which requires a strongly interdisciplinary research approach [20].

Recommendation Algorithms. We exemplified how RE recommendation problems can be solved by conventional recommendation approaches. However, there are other settings with complex inter-dependencies between requirements and a large number of inconsistent stakeholder preferences. These settings require to adapt, combine, and extend existing recommendation approaches. One possible direction is to adapt knowledge-based recommendation functionality for group-based recommendation scenarios, for example, critiquing-based recommendation approaches [6] have to be extended to support different types of group-based recommendation and diagnosis functionalities (for determining repair actions for inconsistent stakeholder preferences).

Quality of Recommendations. Stakeholders are often skeptical regarding a new form of automated tool support. As a consequence, recommendation technologies will only succeed if they deliver high quality recommendations. To this end, we have to design and conduct empirical studies to (a) learn about stakeholder needs and (b) evaluate recommendation systems. The goal is to figure out how existing recommendation algorithms have to be adapted for an optimal performance in RE scenarios. Empirical studies should deliver *grounded theories* about the behavior of stakeholders in particular situations, which are needed to train and optimize related recommendation algorithms.

Social Networks in Recommendation Algorithms. The social status of stakeholders often has enormous impact on RE-related decision processes. Social network analysis is an important supportive technology for different types of recommenders. For example, collaborative filtering recommenders can exploit trust information to improve the quality of item predictions. Group-based recommenders can exploit trust information for determining group recommendations.

Semi-Automated Dependency Detection. Effective dependency management is crucial for efficient requirements engineering processes. Existing recommendation support is focused on the analysis of similarities between requirements (using, e.g., clustering and content-based filtering methods). An important issue for future research is to make dependency detection more intelligent in terms of making it possible to predict, for example, the type of dependency (e.g., refinement or incompatibility dependency). Such new algorithms can rely on concepts from the areas of natural language processing [13] and text mining [46].

Requirements Discovery in Open Source Software Development. Open Source platforms include different types of communication channels and types of communication. As a consequence the filtering of requirement-relevant information is a challenge but a prerequisite for improving the quality of recommendation support. An issue for future research is the development of methods which allow to isolate requirement-relevant artifacts before recommendation algorithms are applied.

Recommendation Beyond Textual Requirements. Existing RE recommendation approaches focus on the analysis of textual requirements specifications which are represented, for example, in a completely informal fashion or in terms of use case scenarios. Future recommendation algorithms for RE should be able to deal with

graphical data sources such as, for example, class diagrams, sequence diagrams, state charts, etc. The inclusion and analysis of such artifacts has the potential to improve the prediction quality of recommendation algorithms and – as a consequence – also to improve the overall efficiency or RE related processes.

Context Awareness There are two basic recommendation modes: *pull* and *push*. *Pulling* means that stakeholders are actively triggering recommendation functionality when needed. When *pushing* the recommender application pro-actively detects situations (contexts) in which a stakeholder needs a particular support [23]. In order to deliver push recommendation, the context of a stakeholder has to be observed and discovered [23]. In RE, detecting the context is difficult to realize. Humans constitute the major part of the working environment and problem situations are implicit, subtle, and subjective. One possible direction is the instrumentation of problem domains to continuously collect users' context and reason about user needs. Contextual recommendation is an emerging field [1] and will also play a major role in the development of recommendation solutions for RE.

5 Conclusions

Due to the increasing size and complexity of software systems as well as the growing share of the degree of distributedness in software projects, recommendation technologies are becoming more and more popular as an intelligent technology for Requirements Engineering (RE). In this paper we focused on a discussion of existing research related to the application of recommendation technologies in different Requirements Engineering scenarios. In order to show the application of recommendation technologies we came up with numerous examples. An outlook provides insights into relevant topics for future research.

Acknowledgements

The work presented in this chapter has been conducted in the IntelliReq (829626) research project funded by the Austrian Research Promotion Agency.

References

1. S. Anand and B. Mobasher. Contextual recommendation. *Discovering and Deploying User and Content Profiles*, pages 142–160, 2007.
2. A. Aurum and C. Wohlin. The fundamental nature of requirements engineering activities as a decision-making process. *Information and Software Technology*, 45(14):945–954, 2003.
3. J. Cleland-Huang B. Mobasher. Recommender systems in requirements engineering. *AI Magazine*, 32(3):81–89, 2011.
4. J. Bettman, M. Luce, and J. Payne. Constructive consumer choice. *Journal of Consumer Research*, 25(3):187–217, 1998.

5. R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69(32):180–200, 2000.
6. R. Burke, A. Felfernig, and M. Goeker. Recommender Systems - An Overview. *AI Magazine*, page to appear, 2011.
7. C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes. In *16th IEEE Intl. Conf. on Req. Engineering (RE'08)*, pages 165–168, Barcelona, Spain, 2008.
8. J. Cleland-Huang, H. Dumitru, C. Duan, and c. Castro-Herrara. Automated support for managing feature requests in open forums. pages 68–74. ACM, 2009.
9. A. Dahlstedt and A. Persson. pages 71–80, Klagenfurt, Austria, 2003.
10. A. Davis. The art of requirements triage. *IEEE Computer*, 36(3):42–49, 2003.
11. C. Duan, P. Laurent, J. Cleland-Huang, and C. Kwiatkowski. Towards automated requirements prioritization and triage. *Requirements Engineering*, 14:73–89, 2009.
12. H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, and C. Castro-Herrara. On-demand feature recommendations derived from mining public product descriptions. pages 181–190, Waikiki, Honolulu, Hawaii, 2011. ACM/IEEE.
13. A. Fantechi and E. Spinicci. A content analysis technique for inconsistency detection in software requirements documents. pages 245–256, Porto, Portugal, 2005.
14. A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *IEEE ICEC'08*, pages 17–26, Innsbruck, Austria, 2008.
15. A. Felfernig, L. Chen, and M. Mandl. Recsys'11 workshop on human decision making in recommender systems. pages 389–390, Chicago, IL, 2005.
16. A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible repairs for inconsistent requirements. In *IJCAI'09*, pages 791–796, Pasadena, CA, 2009.
17. A. Felfernig, W. Maalej, M. Mandl, M. Schubert, and F. Ricci. Recommendation and decision technologies for requirements engineering. In *ICSE 2010 Workshop on Recommender Systems in Software Engineering*, pages 1–5, Cape Town, South Africa.
18. A. Felfernig, M. Schubert, M. Mandl, and P. Ghiardini. Diagnosing inconsistent requirements preferences in distributed software projects. In *3rd International Workshop on Social Software Engineering*, pages 1–8, Paderborn, Germany, 2010.
19. A. Felfernig, M. Schubert, M. Mandl, and P. Ghirardini. Diagnosing inconsistent requirements preferences in distributed software projects. In *3rd International Workshop on Social Software Engineering*, pages 1–8, Paderborn, Germany, 2010.
20. A. Felfernig, C. Zehentner, G. Ninaus, H. Grabner, W. Maalej, D. Pagano, L. Weninger, and F. Reinfrank. Group decision support for requirements negotiation. *Springer Lecture Notes in Computer Science*, (7138):1–12, 2011.
21. C. Fitzgerald, E. Letier, and A. Finkelstein. Early failure prediction in feature request management systems. pages 229–238.
22. J. Golbeck. Computing with social trust. Springer, 2009.
23. Hans-Jörg Happel and Walid Maalej. Potentials and challenges of recommendation systems for software development. In *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. ACM, Nov 2008.
24. H. Hofmann and F. Lehner. Requirements engineering as a success factor in software projects. *IEEE Software*, 18(4):58–66, 2001.
25. J. Iyer and D. Richards. Evaluation framework for tools that manage requirements inconsistency. 2004.
26. A. Jameson, S. Baldes, and T. Kleinbauer. Two methods for enhancing mutual awareness in a group recommender system. In *ACM Intl. Working Conference on Advanced Visual Interfaces*, pages 48–54, Gallipoli, Italy, 2004.
27. U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *19th National Conference on AI (AAAI04)*, pages 167–172, San Jose, CA, 2004.
28. J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news full text. *Comm. of the ACM*, 40(3):77–87, 1997.

29. S. Lichtenstein and P. Slovic. *The Construction of Preference*. Cambridge Univ. Press, 2006.
30. S. Lim and A. Finkelstein. Stakerare: Using social networks and collaborative filtering for large-scale requirements elicitation, in press. *IEEE Trans. on Softw. Eng.*, pages 1–32, 2011.
31. S. Lim, D. Quercia, and A. Finkelstein. Stakenet: Using social networks to analyse the stakeholders of large-scale software projects. pages 295–304, Cape Town, South Africa, 2010. ACM/IEEE.
32. S. Lohmann, T. Riechert, and S. Auer. Collaborative development of knowledge bases in distributed requirements elicitation. In *Software Engineering (Workshops): Agile Knowledge Sharing for Distributed Software Teams*, pages 22–28, 2008.
33. S. Marczak, I. Kwan, and D. Damian. Social networks in the study of collaboration in global software teams. Munich, Germany, 2007.
34. J. Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. *UMUAI*, 14(1):37–85, 2004.
35. J. Masthoff. Group recommender systems. *Recommender Systems Handbook*, pages 677–702, 2011.
36. D. McFadden. Rationality for economists. *Jrnl. of Risk & Uncertainty*, 19(1):73–105, 1999.
37. M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.
38. A. Felfernig R. Burke and M. Goeker. Recommender systems: An overview. *AI Magazine*, 32(3):13–18, 2011.
39. L. Roy R. Mooney. Content-based book recommending using learning for text categorization. *User Modeling and User-Adapted Interaction*, 14(1):37–85, 2004.
40. R. Reiter. A theory of diagnosis from first principles. *AI Journal*, 23(1):57–95, 1987.
41. G. Ruhe, A. Eberlein, and D. Pfahl. Trade-off analysis for requirements selection. *Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 13(4):354–366, 2003.
42. G. Ruhe and M. Saliu. The art and science of software release planning. *IEEE Software*, 22(6):47–53, 2005.
43. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.
44. I. Sommerville. *Software Engineering*. Pearson, 2007.
45. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.
46. I. Witten and E. Frank. *Data Mining*. Elsevier, 2005.