Gerald Ninaus

Recommendation Technologies in Requirements Engineering

Doctoral Thesis

Graz University of Technology

Institute for Software Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Alexander Felfernig
Second reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Martin Pinzger

Graz, February 2016

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

Software project failures (total project failure, cost and schedule overruns, failure to deliver promised functions) lead to additional costs of many billion dollars. Major sources of these failures can be located in the Requirements Engineering phase, which is the first phase of a software development process. The costs of fixing defects in the Requirements Engineering phase are far below the costs in subsequent phases and, as a consequence, improving Requirements Engineering processes can reduce the overall costs of software development dramatically. Although a system that automatically understands user's needs is not yet feasible, semi-automated systems with recommendation techniques can help requirements engineers finding specification failures during the Requirements Engineering phase. Potential techniques are: group recommendations (support the decision making process, for example, during the requirements prioritization process), content-based recommendations (highlight potential relations between requirements), and preference visibility of stakeholders (anonymous vs. non-anonymous).

Within the scope of this thesis a Requirements Engineering web platform with recommendation support is developed and used for several empirical studies conducted at our university. To improve the requirements prioritization process, the impact of preference visibility on the decision making process and the applicability of group recommendation technology in the context of requirements prioritization are evaluated and the results are presented. Furthermore, this thesis introduces three new group recommendation heuristics developed for the use in a requirements prioritization context and shows the improvements compared to well-known group decision heuristics. To increase stakeholder interaction with Requirements Engineering artifacts (requirements and release plans) and to increase stakeholder communication, a traffic light based approach is presented. For the identification of relations between requirements, a similarity measurement approach based on the lexical resource *OpenThesaurus* is introduced and the results of an evaluation of the recommendation quality of dependency candidates are presented. To increase the reuse of requirements, a keyword recommendation technique is introduced and an empirical study clearly shows more reuse in study groups using the keyword recommendation technique. Finally, the application of Human Computation in the context of Requirements Engineering is presented as future work.

Zusammenfassung

Gescheiterte Software Projekte (vollständiger Projektausfall, Kosten- und Terminüberschreitungen, versprochene Funktionen werden nicht geliefert) führen zu zusätzlichen Kosten in Milliardenhöhe. Die Quelle für diese Probleme kann bereits in der Requirements Engineering (Anforderungsanalyse) Phase gefunden werden, welche zu Beginn eines Softwareentwicklungsprozesses steht. Die Kosten für die Behebung von Fehlern in dieser ersten Phase sind erheblich geringer als wenn Fehler in den darauffolgenden Phasen identifiziert und behoben werden müssen. Folglich können Verbesserungen im Requirements Engineering Prozess die Gesamtkosten eines Softwareprojekts erheblich reduzieren. Obwohl Systeme zur automatischen Auswertung von Anforderungen noch nicht realisierbar sind, können halb-automatisierte Systeme durch Verwendung von Empfehlungstechnologien den Anforderungsingenieuren helfen, Spezifikationsfehler bereits während der Requirements Engineering Phase zu finden. Dafür geeignete Technologien sind: Gruppenempfehlung (Unterstützung des Entscheidungsfindungsprozesses, beispielsweise während der Priorisierung von Anforderungen), Content-basierte Empfehlungen (Aufzeigen potentieller Beziehungen zwischen Anforderungen) sowie die Sichtbarkeit von Präferenzen der Stakeholder (anonym vs. nicht-anonym).

Im Zuge dieser Arbeit wurde eine Requirements Engineering Webplattform mit Empfehlungsunterstützung entwickelt. Mit Hilfe dieser Plattform wurden verschiedene empirische Studien an unserer Universität durchgeführt. Um die Priorisierung von Anforderungen zu verbessern wurde die Auswirkung von sichtbaren Präferenzen auf den Entscheidungsfindungsprozess sowie die Anwendbarkeit von Gruppenempfehlungen im Umfeld der Anforderungspriorisierung untersucht und die entsprechenden Ergebnisse präsentiert. Zusätzlich werden in dieser Arbeit drei neue Heuristiken für Gruppenempfehlungen zur Unterstützung von Requirements Engineering Prozessen vorgestellt und mit bestehenden Heuristiken verglichen. Um die Interaktion von Stakeholdern mit Requirements Engineering Artefakten (Anforderungen und Releasepläne) zu erhöhen und um die Kommunikation zwischen Stakeholdern zu steigern wird ein ampelbasierter Ansatz präsentiert. Für die Identifizierung von Relationen zwischen Anforderungen wurde eine Methode zur Ähnlichkeitsmessung basierend auf der lexikalen Ressource *OpenThesaurus* eingeführt. Darauf aufbauend wird eine Evaluierung der Empfehlungsqualität von Kandidaten für mögliche Relationen zwischen Anforderungen präsentiert. Um die Wiederverwendung von Anforderungen zu erhöhen

wird eine Stichwort-Empfehlungstechnik vorgestellt und es wird gezeigt, dass Gruppen mit dieser Empfehlungstechnik häufiger Anforderungen wiederverwenden. Abschließend wird die Anwendung von Human Computation im Umfeld von Requirements Engineering als Future Work vorgestellt.

Acknowledgment

First and foremost, I would like to thank my supervisor Univ.-Prof. Dipl.-Ing. Dr.techn. Alexander Felfernig for his continuous support, discussions, his motivation, and valuable comments throughout the preparation of this thesis.

I also want to acknowledge and thank my colleagues Dipl.-Ing Florian Reinfrank, Dipl.-Ing. Harald Grabner, Michael Jeran, Dipl.-Ing. Stefan Reiterer, Dipl.-Ing. Martin Stettinger, Dipl.-Ing. Klaus Isak, Assoc.Prof. Mag. Dr. Gerhard Leitner, Dipl.-Ing. Dr.techn. Monika Mandl, Dipl.-Ing. Christoph Zehentner, Petra Pichler, and Arabella Gaß for their support. In addition, I want to thank our industrial partners, Dipl.-Ing. Walter Schanil and Dipl.-Ing. Leopold Weninger, for many fruitful discussions and valuable input to my work.

I am also very grateful to the woman beside me, Martina, for her understanding and support while I was writing my thesis. Last but not least, I want to thank my family for the support and belief in me during the years of study.

Gerald Ninaus
Graz, 2016

Contents

List of Figures

1 Introduction

Nowadays, software systems have a tremendous impact on business and our society. For example, if someone retrieves cash from an ATM, is doing a phone call, or is driving a car, every time software is involved. Similar to the social impact, investments in information technologies increase and are now among the largest expenses [Cha05].

Although software solutions are crucial for many business cases, there is a huge amount of projects which fail. The definition of the term *project failure* itself is controversial. There are many different and often contradictionary definitions of project failure in the literature indicating a lack of consensus [PJ90]. One accepted approach is to define a failed project as any project which has been canceled (total project failure) or did not meet its budget (cost overruns), delivery (schedule overruns), and/or business objectives (delivered fewer or wrong functions). Based on this broad approach several studies showed that more than 70% of all software projects fit into this failure definition [WK04][Lin99]. With the huge impact of software on business and our society, these failures lead to additional costs of many billion dollars [Cha05].

Among other reasons, *unrealistic or not articulated project goals*, *misunderstanding of requirements*, *inaccurate estimates of needed resources*, *badly defined system requirements*, and *stakeholder politics* (e.g. irrational reasoning based on emotions) can be considered as major source for project failures [Cha05][BP84].

All sources of project failures mentioned above are related to the Requirements Engineering phase, which is the first phase of a software development process. Requirements Engineering can be considered as the branch of software engineering concerned with the real-world goals for functions of and constraints on software systems with requirements defined as a representation of decision alternatives or commitments [AW03][Zav97]. Decisions made in the Requirements Engineering phase can be considered as critical as they are the basis for all subsequent tasks in the development process and high quality requirements are a major precondition for the success of projects [Fel+12]. To show the potential impact of the Requirements Engineering process on the software project, a recent Gartner report [Gar11] states that *requirements defects are the third source of product defects (following coding and design), but are the first source of delivered defects. The cost of fixing defects ranges from a low of approximately $70 (cost to fix a defect at the requirements phase) to a*

high of $14.000 (cost to fix a defect in production). Improving the requirements gathering process can reduce the overall cost of software and dramatically improve time to market. Consequently, poorly implemented Requirements Engineering is one major risk for project failure [Yan+08][HL01]. Despite the critical impact of the Requirements Engineering process on the success of projects and the fact that project failure is in most cases predictable and avoidable, rarely more than 2-4% of the overall project efforts are spent for Requirements Engineering [Fir04][Cha05].

Requirements Engineering, on the other hand, is a difficult task which needs a lot of attention and resources spent. First, requirements analysts start with ill-defined and in many cases conflicting ideas of what the proposed system is expected to do and must consider the needs of users, customers, and other stakeholders [CA07]. Second, the amount of knowledge and number of stakeholders involved in Requirements Engineering processes tend to increase making individual as well as group decisions much more difficult. Additionally, with an increased number of attributes (e.g. various platforms, different types of users) concerning software projects the amount of requirements that must be satisfied by project teams increases as well [RSW08]. Thus, requirements analysts are often faced with situations where the amount and complexity of requirements outstrips their capability to survey them and to reach decisions [Dav03].

Low resources and increasing complexity lead to the logical conclusion that there exists a need for more advanced tools to assist Requirements Engineering processes. This assumption is also supported by the results of an empirical market research done in the year 2004 that summarizes the request to improve Requirements Engineering tasks by the use of Computer Aided Requirements Engineering (CARE) [LMP04][MKF14].

Although the prospect of a support system that would automatically understand user's needs is very appealing, this is not yet feasible, especially due to the fact that requirements are normally defined in natural language as it cannot be assumed that all stakeholders have the necessary knowledge to use formal descriptions [CA07]. However, less sophisticated systems can sufficiently support the work of requirements engineers, for example, by preprocessing the provided data, by facilitating the communication, and by supporting decision making in groups [Rya93]. These and similar functionalities can be provided by the use of recommender systems, which 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* [Bur02].

To be more specific, group recommendations can be used to support stakeholders in the decision making process. For example, they can facilitate the finding of a consensus between stakeholders [FN12][Nin12][Nin+14b]. Content-based recommender systems can be used to process requirements descriptions and metadata to highlight potential relations between

requirements [Nin+14a]. Additionally, collaborative recommenders can be used to help stakeholders navigating through large sets of different requirements and guide them to artifacts which are likely to be of interest for them [Cas+08]. Finally, enhancing recommendation techniques with elements of Human Computation [Ahn05] (micro contributions provided by stakeholders for tasks computer cannot fulfill) can be used to increase the quality of the support of CARE systems as well.

1.1 Research Objectives

In this section the research objectives of this thesis are discussed - these objectives are related to three main categories.

1. Investigation of the effects of recommender systems applied to requirements negotiation tasks and how the visualization of preferences of other group members influence decision making. These findings are used as a basis for our INTELLIREQ prototype, which is a CARE environment that includes different recommendation techniques.

2. Development of the INTELLIREQ prototype to evaluate the impact of recommender systems in the context of Requirements Engineering processes with a focus on four different types of decisions: *quality decisions, preference decisions, classification decisions*, and *property decisions* [Reg+01]. For the evaluation of the impact, several empirical studies are conducted.

3. Discussion of concepts to enhance recommendation techniques by using Human Computation for tasks computers are not able to fulfill. For example, identifying the correct context of a term in a requirements description is not an easy task for a computer system but can be easily done by humans.

1.1.1 Human Factors in Requirements Engineering

For the support of requirements negotiation scenarios, group recommendation systems can be used which provide functionalities such as the visualization of preferences of other group members, recommendations for individual and group decisions, and recommendations for conflict resolutions in case of inconsistent stakeholder preferences [Mas11][JBK04]. In this context it is necessary to pay special attention on how human decision making is done.

In the economic world there exists a standard model which tries to describe the behavior of consumers *as if information is processed to form perceptions and beliefs using strict*

Bayesian statistical principles (perception-rationality), preferences are primitive, consistent, and immutable (preference-rationality), and the cognitive process is simply preference maximization, given market constraints (process-rationality) [McF99]. However, decision models based on this rational thinking are not applicable in most Requirements Engineering scenarios for two reasons: first, stakeholders do not exactly know their preferences beforehand and need to develop them during the decision making process. Second, preferences of stakeholders are not stable during requirements development and negotiation which is also conflicting with the aforementioned idea of rational thinking [AP05][BJP98].

It is therefore necessary to use incremental and adaptive preference acquisition approaches in Requirements Engineering systems such as recommendation technologies taking into account the aspect of preference construction [PC08][Bly02]. For example, group recommendation techniques need to be evaluated in the context of incremental preference elicitation scenarios where stakeholders first define personal preferences and, afterwards, make final group decisions. For the measurement of the applicability of recommendation technologies in the context of Requirements Engineering the impact of preference visibility and group recommendation technologies on the dimensions *usability* and the *quality of decision support* need to be evaluated.

(Q1) *How do recommendations and preference visibility influence the perceived usability and quality of decision support in Requirements Engineering environments?*

1.1.2 Improvement of Decision Tasks

Based on the findings of our study regarding the impact of recommender systems on the perceived *usability* and *quality of decision support* in Requirements Engineering environments (see Chapter 3) it is necessary to evaluate concrete techniques for the enhancement of CARE systems. Clearly, decision-making in Requirements Engineering processes differs in some aspects from other decision-making scenarios such as the purchasing of movies. However, there are other aspects which are in general valid and can, at least partially, be explained with classical decision-making [Reg+01]. It is therefore a good advice to start the development of recommendation techniques for Requirements Engineering scenarios by using existing knowledge gathered in the field of recommender systems. The remainder of this section describes four decision types in the context of Requirements Engineering which we want to support with recommendation technologies.

Quality decisions are made to estimate if a requirement is concrete and understandable which is a necessary precondition for a subsequent requirements negotiation phase. It is also necessary to create a document at the end of a Requirements Engineering process

which contains a common agreement by all involved stakeholders on the final specification [Poh13]. However, requirements are often defined in natural language which is in many cases ambiguous because the meaning of a word can depend on the context it is used in. This often leads to misinterpretations by different stakeholders [Fer+14]. Another source of error is tacit knowledge in terms of, for example, implicit stakeholder goals, hidden assumptions, and unshared expectations [GB01]. Additionally, it is often not an easy task to estimate the completeness of requirements and to find the will-power to do the tasks necessary to make a requirement complete. Requirements completeness, however, is important because it affects customer acceptance (users can reject to use software because of missing features), development costs (can be underestimated), and verification (incomplete requirements are ambiguous) [Fir05]. Furthermore, a sound release plan is necessary to avoid difficulties in the subsequent development process. For example, having too many tasks assigned to a single release will force the developer to do overtime and / or spend insufficient time for the implementation of single requirements, which will, most probably, lead to higher costs and lower quality of the implemented source code.

As a consequence, it is necessary to improve the interaction with requirements, for example, by assuring that a sufficient number of stakeholders participated in the quality assurance of requirements descriptions. For release plans, an improvement in the interaction can be achieved by indicating that too many or too less hours have been assigned to a specific release.

(Q2.1) *How to increase stakeholder interaction with requirements and release plans to improve the quality of these artifacts?*

Property decisions address the estimation of certain properties attached to requirements. Examples of this additional information are the *implementation costs* of a specific requirement or whether a requirement cannot be satisfied without the implementation of other requirements [Reg+01]. These properties can be defined as metadata of requirements and are described as data about data. Thus, for the definition of a complete *individual* requirement it is necessary to ensure that missing mandatory metadata has been filled in [Fir05] and that this information is based on the distributed knowledge of as many involved stakeholders as possible. For example, if we consider the *implementation cost* property of a requirement, missing information about the technical feasibility or the customers needs can lead to critical misinterpretations which result in wrong cost estimations. To discover a gap in the distribution of knowledge, the preferences of stakeholders can be compared. If there is a high dissent with respect to property values this indicates a lack of distributed knowledge. As simple example, if someone defines a *User Login* function as a requirement, one could assume that this requirement only covers a simple interaction interface. Another one could assume, that this also includes necessary security actions. This will probably result in a dissent about the necessary effort to implement the requirement.

For a complete and sound release plan it is also necessary to identify dependency relations between requirements, which are considered as property decisions as well. This information is important as it facilitates the identification of inconsistencies between requirement descriptions and helps to identify redundant information in the set of requirements. As the identification of dependencies is a labor-intensive task, a recommendation of possible dependency candidates is of high interest [LMP04][CA07]. We therefore want to investigate techniques to calculate the most interesting relations between requirement descriptions which can be used as recommendation to support stakeholders in dependency detection and definition.

(Q2.2) *How to increase stakeholder communication about property decisions?*
(Q2.3) *How to identify candidates for dependency relations between requirement descriptions?*

Classification decisions define to which topic or keyword a requirement belongs to. Obviously, any succeeding process benefits from requirements classification as it facilitates the navigation through the set of requirements. Additionally, a major exploitation of requirements classification can be found in reuse. Reuse can be seen as better resource utilization because it reduces the overall development effort. This applies not only to the creation of requirement documents but also to the attached software artifacts which have been developed in earlier projects. Also, it is a source to identify difficulties which can occur with a certain requirement due to the exploitation of knowledge from the past [CR00]. For example, a complex search query cannot be executed within a desired response time on the used hardware and / or with a certain database system.

Unfortunately, requirements reuse is often a complex task due to the fact that requirements are written in natural language which circumvent effective reuse. To overcome this problem, requirements need to be processed in a special way to make their organization into repositories of reusable artifacts effective [CR00]. One possible approach is to derive useful keywords from the descriptions of requirements. However, the identification of important terms in a description written in natural language is not an easy task for computer systems as this process needs knowledge about the domain and the context in which these terms are used. Fortunately, this kind of needed knowledge is possessed by stakeholders working on the requirements definition. Therefore, a semi-automatized approach helping stakeholders defining keywords for requirements in a central repository appears to be promising. With this support the retrieval of useful requirements can be improved and will, as a consequence, increase the reuse of requirements.

(Q2.4) *How to increase the reuse of requirements in software projects?*

Preference decisions deal with the prioritization of requirements and the succeeding definition whether a requirement should be considered for the next release or not. This process is of exceptional importance as it is the only way to ensure the implementation of the most important requirements in software projects which are confronted with resource limitations [HL01][Wie99]. Properly prioritizing leads to significant benefits such as *improved customer satisfaction*, *lower risk of cancellation*, *force stakeholders to address all requirements and not just their own*, and systematically *prioritize investments* (e.g. allocate limited resources for quality assurance only for the most important requirements) [Fir04]. However, finding consensus between stakeholder preferences regarding the prioritization is a challenging task as every stakeholder holds her / his own point of view towards the software project. There are also psychological effects which influence the decision making process such as the cognitive dissonance [Fes57], group polarization [ZCW92], and the primacy effect [Fel+07]. Consequently, it is necessary to introduce a tool-supported prioritization, for example, by using recommendations based on stakeholder preferences, to reduce the influence from such psychological effects [Wie99].

Preference decisions can be supported by the use of group recommendation technologies which provide functionalities such as the visualization of preferences of other group members and recommendations for group decisions. For the improvement of support functionalities in CARE systems it is necessary to evaluate the impact of preference visibility on the quality of the Requirements Engineering process. With this knowledge it is possible to define effective visualization techniques for CARE systems to increase the quality of Requirements Engineering processes.

To make effective use of recommendations in group decision making it is necessary to identify group recommendation heuristics with high predictive quality (i.e. try to estimate decisions based on the preferences of stakeholders). Consequently, an evaluation of different group recommendation heuristics is needed in the context of requirements negotiation scenarios to provide the background for the development of useful recommendation strategies for CARE systems.

(Q2.5) *Does preference visibility influence the quality of the Requirements Engineering process?*
(Q2.6) *What is the prediction quality of group recommendation heuristics in a Requirements Engineering scenario?*

1.1.3 Human Computation

Although the capabilities of computer systems are vastly increasing, there are still tasks which can only be accomplished by humans. For example, CAPTCHAs (Completely Automated Public Turing Test to Tell Computers and Humans Apart) include tasks in which humans are superior to a computer [Ahn05]. For example, the identification of pictures which contain image distortions is still challenging for computer systems while humans can complete this task more successfully. CAPTCHAs are used to discriminate between human users and bots interacting with an interface [Ahn05]. However, Human Computation can be utilized to gather useful input from human interaction. In the year 2005, von Ahn introduced the *ESP* game with the purpose to label images found on the internet. The game was played by a total of 13,630 people who generated 1,271,451 labels for 293,760 different images [Ahn05]. It is notable that the only motivation for the players to participate in the game was having fun. The resulting quality of the labels for the images outperformed algorithms in the field of computer vision known at that time [Ahn05].

With the huge amount of information written in natural language during Requirements Engineering processes, a text analysis is a prominent approach to support requirements engineers during the task of defining complete and error-free requirement documents. However, text analysis struggles with some difficulties: first, words can have a different meaning depending on the context they are used in. Second, there are important terms only used within a company context and cannot be inferred from general purpose ontologies. Also, having a database containing the vast amount of information necessary to achieve human-like reasoning is implausible. For example, Marvin Minsky, a founder of AI, once estimated that knowledge about 30 to 60 million things and the power to make analogies based on them is necessary for commonsense reasoning [LS04]. Hence, integrating *Human Computation* into the natural language process seems to be promising if it is possible to combine the fast processing power of computer systems with the commonsense reasoning capabilities of humans.

(Q3) *How to use the idea of Human Computation to support Natural Language Processing in the context of Requirements Engineering?*

1.2 Contributions

The major contribution of this thesis is the improvement of Computer Aided Requirements Engineering (CARE). In this context we investigate different aspects which can be supported

by recommender technologies. Table 1.1 provides an overview of the research questions and the corresponding contributions.

Table 1.1: Overview of the research questions and the corresponding contributions.

Research Questions	**Contributions**
(Q1) How do recommendations and preference visibility influence the perceived usability and quality of decision support in Requirements Engineering environments?	To answer this research question we present the results of a study [Fel+12]. We used our INTELLIREQ environment (see Section 3.2) to support 293 software developers with the task of defining a requirements set for development in a software project. To identify the effect of group recommendation and preference visibility we created different study groups with and without visible preferences and / or group recommendations. After the interaction with the system the study participants were asked to fill out a questionnaire about the perceived usability and the perceived quality of decision support of the used system. We present the results of our evaluation in Section 3.3.3.
(Q2.1) How to increase stakeholder interaction with requirements and release plans to improve the quality of these artifacts?	To increase the interaction with requirements and release plans with flaws in their specifications we introduced a traffic light based guidance in our INTELLIREQ environment (see Section 6.3). With this technique we pointed stakeholders to, for example, requirements which have not been visited from a certain amount of stakeholders (defined by a threshold). We also used this traffic light indicator for flaws in the release plan if the assigned amount of requirements exceeded the available resources or if there were not enough requirements assigned to releases with more available resources. We evaluated the impact of the traffic light based guidance in a study [Nin+14b] by comparing user interaction during Requirements Engineering tasks and present the results in Section 6.4.

Table 1.1: Overview of the research questions and the corresponding contributions.

Research Questions	Contributions
(Q2.2) How to increase stakeholder communication about property decisions?	To increase stakeholder communication, we introduced a metadata interaction interface in our IntelliReq environment (see Section 6.3). Within this interface stakeholders were able to assign properties to a requirement. To stimulate discussion between stakeholders we marked potential problems with a traffic light annotation. For example, if there is a dissent between the involved stakeholders about the implementation costs of a specific requirement, this event is marked with either a yellow or red traffic light. In our notion, a slight dissent was marked with yellow, while a larger dissent was marked with a red light. For evaluation purposes, we conducted a study [Nin+14b] where participants were asked to define a requirements document by using our system. In this study we investigated the impact of the extended user guidance with traffic lights on the efficiency of the requirements engineering task; we present our results in this thesis.
(Q2.3) How to identify candidates for dependency relations between requirement descriptions?	To generate a recommendation of dependency candidates we developed an approach based on the lexical resource *OpenThesaurus*[1] and present it in Section 4.4.2. For evaluation purposes we generated a test set of 30 different requirements and used our recommendation technique to calculate a set of candidates for dependencies. In a study we conducted at our university, participants were asked to evaluate the quality of these recommendations; related results of the study are presented in Section 4.4.2.

[1] http://www.openthesaurus.de

Table 1.1: Overview of the research questions and the corresponding contributions.

Research Questions	Contributions
(Q2.4) How to increase the reuse of requirements in software projects?	To evaluate the possibility to increase the reuse of requirements we conducted a study [Nin+14a]. The participants were confronted with the task to generate a requirements set for a software application. To provide the possibility for reuse we created a central storage (repository) for requirements which was accessibly by all teams within a study group. Hence, participants first tried to identify candidates for reuse in the repository. If their search was without success, the participants created a new requirement which was automatically stored into the central storage. For browsing through the requirements repository the participants could select a specific keyword from a given list and received all connected requirements for the specific keyword. For our evaluation we used two study groups: one was supported by our keyword recommender which proposed candidate terms for the annotation of requirements based on the description provided during the creation process of the requirements. The second group was not supported by the keyword recommender but there was also the possibility to browse through an own repository generated for the second study group to find requirements for reuse. In this thesis we present our evaluation of the results (see Section 4.5).
(Q2.5) Does preference visibility influence the quality of the Requirements Engineering process?	For the evaluation of the influence of preference visibility we evaluated the development process of 39 software teams [NFR12]. In this study, software development teams had to create a requirements document for a software application. In a succeeding step these teams had to implement all defined requirements. For our evaluation we divided the software teams into two study groups. In the first group, participating teams could see the real name of team members associated to their preferences for requirements. In the second group, real names were replaced by placeholders like user 1 or user 2. To identify the impact of visible real names associated to preferences of team members we evaluated the quality of the final software products programmed by the different teams and present the results in Section 5.3.

Table 1.1: Overview of the research questions and the corresponding contributions.

Research Questions	Contributions
(Q2.6) What is the prediction quality of group recommendation heuristics in a Requirements Engineering scenario?	For the evaluation of the prediction quality we used a data set collected during our study [Nin12]. This data set consisted of preferences of team members and the corresponding decisions taken by the teams. In a first step, we used already well known decision heuristics to calculate predictions for these decisions based on the team member preferences. In a next step we created three new heuristics: *Median Based*, *Ensemble Based*, and *Standard Deviation Based*. These new heuristics were compared to the existing heuristics with regard to prediction quality (see Section 5.4).
(Q3) How to use the idea of Human Computation to support Natural Language Processing in the context of Requirements Engineering?	In Chapter 7 we discuss ideas on how to adopt the games originally presented by von Ahn [Ahn05] to fit into a Requirements Engineering process. We therefore focus on two main problem areas – unrecognized ambiguity and the need to define relations between terms in an area of interest during the interpretation of requirement descriptions. To cope with unrecognized ambiguity we present a modified version of the *ESP* game introduced by [Ahn05]. We describe necessary alterations to the game due to the different task and context. For the definition of relations of objects we propose a game to identify relations. The implementation and evaluation of these ideas is not part of this thesis but should inspire future work.

1.3 Thesis Outline

In this thesis, related and future work are included in the specific chapters. This thesis consists of eight chapters, which are organized as follows:

Chapter 1 introduces the motivation and research objectives of this thesis. Finally, an overview of the structure of this thesis concludes the chapter.

Chapter 2 gives a detailed overview of *Recommender Systems in Requirements Engineering*. In Section 2.2 we discuss existing research dedicated to the application of recommendation technologies in Requirements Engineering. Section 2.3 is dedicated to potential basic application scenarios in which recommendation technologies can be exploited. In Section 2.4 we

discuss issues for further research based on our analysis.

Chapter 3 analyzes the impact of applying group recommendation technologies to improve the quality of decision processes in the context of *requirements negotiation* which is the process of resolving existing conflicts between stakeholder preferences and deciding which requirements should be implemented. In Section 3.2 we describe the software platform used for our study. This section also includes hypotheses and results of the conducted study. In Section 3.4 we summarize related work.

Chapter 4 is about recommending requirements for reuse and detecting dependency candidates between requirements. In Section 4.2 we give a brief overview of document representation, word senses, and domain knowledge. Section 4.3 gives a short overview of IntelliReq and the used lexical resource. In Section 4.4 we introduce two content-based recommendation techniques and present the conducted study with related results.

Chapter 5 summarizes our work done in the field of recommending prioritizations of requirements. In Section 5.2 we introduce the study settings used for our evaluation. In Section 5.3 we discuss anonymous preference elicitation. In this context we investigate the influence of anonymous preferences on different dimensions such as the consensus in project teams, decision diversity, and the overall output quality of software projects. In Section 5.4 we compare well known group decision heuristics with decisions of software development teams. We also introduce three new heuristics and compare them with state of the art heuristics.

Chapter 6 covers our final version of the IntelliReq prototype. In Section 6.2 we discuss recommendation techniques used in the IntelliReq prototype. Section 6.3 introduces the user interface of IntelliReq. We describe our traffic light annotation and introduce the dependency view. Section 6.4 shows the benefits of our implementation and presents the results of our evaluation of the IntelliReq prototype. In Section 6.5 we focus on topics for future work related to recommender systems in the context of IntelliReq.

Chapter 7 introduces ideas for the usage of Human Computation to enhance the Requirements Engineering process. In Section 7.1 we discuss the problem of unrecognized ambiguity and introduce an altered version of the *ESP* game originally introduced by Ahn [Ahn05]. In Section 7.2 we discuss another game variant to identify relations between objects in a specific area of interest.

Chapter 8 concludes this thesis. We reflect on our research questions and contributions.

2 Overview of Recommender Systems in Requirements Engineering

This chapter is based on the work published in [Fel+13]. The author of this thesis contributed an *in-depth literature analysis* and wrote *major parts of the above mentioned paper*.

2.1 Introduction

Core activities of a Requirements Engineering process are elicitation & definition, quality assurance, negotiation, and release planning [Som11]. Due to the increasing size and complexity of software systems, we can observe a growing demand for intelligent methods, techniques, and tools that can help to improve the overall quality of Requirements Engineering processes [MC11][Fel+10b][MT09]. In this chapter we focus on the aspect of how different types of recommendation technologies [BFG11] can be applied to support stakeholders in the completion of their Requirements Engineering tasks.

Recommender systems are intensively applied for the purpose of recommending products and services such as movies, books, digital cameras, and financial services. 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* [Bur00][BFG11]. Such systems support users in the identification of relevant items in situations where the amount and/or complexity of an assortment outstrips their capability to survey it and to reach a decision [Bur02]. *Low-involvement items* such as movies and books are often recommended by analyzing the preferences of users with a similar rating behavior. The corresponding recommendation approach is collaborative filtering [Her+04] which is a basic implementation of word-of-mouth promotion where purchase decisions are influenced by the opinion of relatives and friends: if two users rated similar items in a similar fashion in the past, a collaborative filtering based recommender systems would propose new items to one customer that the other one has already rated positively. The online selling platform amazon.com recommends items which have already been purchased by customers with a rating behavior similar to that of the current customer [LSY03]. An alternative approach to the recommendation of low-involvement items is content-based filtering [PB97]. It is an approach

to information filtering where item features a user preferred in the past are exploited for determining new recommendations. For example, if a customer of amazon.com bought books related to the Linux operating system, similar books (related to Linux) will be proposed in future recommendation sessions. *High-involvement items* such as digital cameras or financial services are recommended on the basis of knowledge-based recommender applications where predefined recommendation rules are exploited by the recommendation engine to determine a set of candidate items [FB08]. Rating-based recommendation approaches are not applicable to high-involvement items since such items are not purchased frequently and therefore no up-to-date rating data is available.

Our major contributions in this chapter are the following. First, we provide an overview of research related to the application of recommendation technologies in Requirements Engineering. Second, we show in detail how different types of recommendation techniques can be applied to proactively support users in different types of Requirements Engineering scenarios. Third, we want to stimulate new ideas and research activities by discussing a couple of issues for future work.

The remainder of this chapter is organized as follows. In Section 2.2 we provide an overview of existing research on the application of recommendation technologies in Requirements Engineering. In the following we discuss different application scenarios for recommendation technologies with a focus on collaborative filtering [Kon+97], content-based filtering [PB97], clustering [WF05], knowledge-based recommendation [Bur00][FB08], group-based recommendation [Mas04], and social network analysis [Gol09] (Section 2.3). Relevant issues for future research are discussed in Section 2.4. With Section 2.5 we conclude this chapter.

2.2 Research on Recommender Systems in Requirements Engineering

In this section we discuss existing research dedicated to the application of recommendation technologies in Requirements Engineering. Our discussion of related research is organized along the typical activities in a Requirements Engineering process. In this context, we take into account the activities of *requirements elicitation & definition*, *quality assurance*, and *negotiation & release planning*. For each activity we discuss relevant application scenarios for recommendation technologies identified in the literature. Table 2.1 provides an overview of these scenarios. To provide further technical insights into the recommendation approaches discussed in this chapter, we present examples of their application in Section 2.3.

Table 2.1: Overview of recommendation approaches for Requirements Engineering.

Activity	Scenario S_i	Recommendation Approach	
elicitation & definition	recommending stakeholders (S_1)	social network analysis	Lim et al. [LQF10]
		content-based filtering	Castro et al. [Cas+08] Cleland-Huang et al. [MC11]
	recommending requirements (S_2)	content-based filtering	Dumitru et al. [Dum+11]
		social network analysis	Lim and Finkelstein [LF12]
		collaborative filtering	Castro et al. [Cas+08] Cleland-Huang et al. [MC11]
quality assurance	managing feature requests (S_3)	clustering	Cleland-Huang et al. [Cle+09]
		machine learning	Fitzgerald et al. [FLF11]
	consistency management (S_4)	knowledge-based recommendation	Felfernig et al. [Fel+10a]
	dependency detection (S_5)	clustering	Cleland-Huang et al. [Cle+09]
negotiation & planning	triage (S_6)	clustering utility theory	Duan et al. [Dua+09]
	release planning (S_7)	group recommendation utility theory	Felfernig et al. [Fel+12]

2.2.1 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 (S_1). This is an important task in the early phases of a Requirements Engineering process since, for example, a low degree of user involvement in most cases leads to project failure [LQF10]. 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 an important and challenging task [MC11][LQF10]. A common mistake is that wrong representatives of groups are integrated into a project or that important stakeholders are simply omitted.

StakeNet [LQF10] is an approach to stakeholder identification which is based on the concepts of social network analysis [Gol09][MKD07]. In *StakeNet*, an initial set of stakeholders and recommendation information provided by these stakeholders is applied for the construction of a social network (SN). 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 techniques 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 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. [MC11] and Castro-Herrera et al. [Cas+08] show how to exploit clustering techniques for grouping user requirements and in the following to assign (recommend) stakeholders to clusters on the basis of content-based filtering [PB97]. 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 (S_2). 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. [Dum+11]. The basic idea is to analyze 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 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 [LF12] introduce the *StakeRare* approach which supports the identification and reuse of requirements. *StakeRare* [LF12] is based on the aforementioned *StakeNet* approach [LQF10]. 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 [Kon+97]. 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 approaches discussed in Cleland-Huang et al. [MC11] and Castro-Herrera et al. [Cas+08] also support 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 probability 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.

2.2.2 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 (S_3). 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 irrelevant features [MC11]. An approach to support effective feature management has been introduced by Cleland-Huang et al. [Cle+09] where clusters of similar requirements are exploited for the identification of redundancies and the prioritization of feature requests. Fitzgerald et al. [FLF11] introduce an approach to feature request management which is based on the idea of predicting software failures (e.g., abandoned 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 techniques for the construction of a prediction model for failures. The basis for learning this prediction model are logged 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 (S_4). Inconsistencies between requirements are resulting from factors such as not enough time for consistency checking, different perceptions and goals, or different granularity of knowledge [IR04]. Especially for informally defined requirements the complete automation of consistency management is unrealistic [IR04] but semi-automated tools help to keep the efforts acceptable. Assuming the existence of a formal description of the requirements model (on the basis of a constraint satisfaction problem [Tsa93]) and the stakeholder preferences (priorities) regarding the defined set of requirements, Felfernig et al. [Fel+10a] 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 [Rei87].

Dependency Detection (S_5). Due to the fact that requirements are often represented on an informal level, 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. 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. [Cle+09]). 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 [FS05].

2.2.3 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 [REP03].

Requirements Triage (S_6). 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 be 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 [AW03] which has to take into account different soft factors such as company policies, personal preferences, and social relationships between stakeholders. The term *requirements triage* stems from medical decision making [Dav03]. In disaster scenarios victims are categorized into three 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. [Dua+09] 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. This utility function is based on the number of clusters a requirement is included in and the weights of these clusters.

Recommendation of Release Plans (S_7). Ruhe et al. [RS05] introduce an approach to release planning that is based on the concept of linear programming [Sch98]. The basic idea is to define a linear program that should calculate a sequence of assignments of features to a corresponding release taking into account the dependencies between the different features. Ruhe et al. [REP03] show how to apply AHP (Analytical Hierarchy Process) for determining a set of preferred requirements. Felfernig et al. [Fel+09][Fel+10a] extend the work of Ruhe et al. [REP03] 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.

These are important contributions to improve the quality of requirements selection but depend on the assumption that stakeholders know their preferences and that *preferences remain stable*. Traditional models of human decision-making are based on the assumption that humans are taking decisions on the basis of rational thinking [McF99]. Following these models, 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 [BJP98]. 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/his mind and significantly increase the upper limit of the price. This typical example of preference reversal [LS06] indicates the non-existence of stable preferences. Instead, the model of preference construction [BJP98] 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 techniques in Requirements Engineering is to exploit basic *decision heuristics* [Mas11] 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 [Mas11]. Felfernig et al. [Fel+12] applied group decision heuristics in the context of Requirements Engineering 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.

2.3 Recommendation Techniques for Requirements Engineering

In order to show how recommendation technologies can be exploited in the Requirements Engineering 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. Information does not substitute communication, i.e., effective Requirements Engineering 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 Requirements Engineering 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 Requirements Engineering application scenarios for the major types of recommendation technologies which are *content-based filtering* (CBF) [PB97], *clustering* [WF05], *collaborative filtering* (CF) [Kon+97], *group recommendation* (GR) [JBK04], *social network analysis* [Gol09], and *knowledge-based recommendation* (KBR) [Bur00][FB08].

Content-based Filtering. Content-based filtering (CBF) [PB97] 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 2.2). 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 [MR00] and the determination of similar requirements can

Table 2.2: Example of a content-based filtering recommendation problem.

requirement	category	planned release	efforts (person days)	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

then be based on the similarity of the extracted keywords – a simple corresponding similarity metric is shown in Formula 2.1.[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*}.

$$sim(s,r) = \frac{|keywords(s) \cap keywords(r)|}{|keywords(s) \cup keywords(r)|} \tag{2.1}$$

k-Means Clustering. A basic method for determining clusters is *k-means clustering* [CO90] 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 [WF05] – for the purposes of our example we apply the similarity between keywords (see Table 2.3) extracted from the textual description of our example requirements ($\{r_1, r_2, r_3, r_4\}$ in Table 2.2). 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) [Kon+97] is perhaps the most widespread recommendation approach where information about the rating behavior 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

[1] Note that the parameter s in Formula 2.1 represents a *user profile*; however, this approach can as well be applied to calculate the similarities between different requirements, i.e., $sim(r_i, r_j)$.

Table 2.3: Keywords extracted from the textual requirement descriptions in Table 2.2.

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

by CF recommenders are of the form *users who were interested in item A were also interested in item C*.

Table 2.4: 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 2.4 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 [Kon+97]. User-based CF tries to identify the k-nearest neighbors (stakeholders interested in a similar set of requirements) of the current user (stakeholder) and calculates a prediction for the rating of 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, collaborative filtering would recommend

requirement r_4 to stakeholder s_1 since the nearest neighbor of s_1 has already viewed r_4.

Group Recommendation. The major goal of group recommendation (GR) technologies [JBK04] is to support/achieve consensus among group members. GR 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 [JBK04]). GR includes 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. 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 accepted without a change. 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 2.5.

Table 2.5: 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 (person days)	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 [Mas04], 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 *person 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 [Mas04].

Social Network Analysis. With the concepts of Social Network Analysis different properties of a network of stakeholders engaged in a Requirements Engineering process can be identified. In order to sketch the analysis of *betweenness centrality* of stakeholders, we

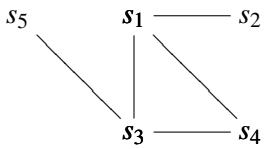

Figure 2.1: Example Social Network (SN) derived from communication patterns of Table 2.6.

introduce the communication patterns between the stakeholders $\{s_1, s_2, s_3, s_4, s_5\}$ in Table 2.6. 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 2.1) 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 2.7 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 (both are part of 3 shortest paths between stakeholders s_j ($s_j \neq s_i$)) whereas the other ones have a betweenness centrality of 0.0. Betweenness centrality reflects the role of a person in communication processes (maybe also related to certain topics). As such, this measure can be exploited as a first basic selection criteria for stakeholders who should participate in a project.

The measure reflects the number of people to whom a person is connecting indirectly through their direct links

Table 2.6: 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

Knowledge-based Recommendation. Knowledge-based recommendation (KBR) [Bur00][FB08] exploits formal 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 in terms of rules (constraints) allows the generation

Table 2.7: *Betweenness Centrality* values for stakeholders $\{s_1, s_2, s_3, s_4, s_5\}$. For example, stakeholder s_1 has a betweenness centrality of 3 since he/she is included in 3 shortest paths between stakeholders s_j ($s_j \neq s_i$); these shortest paths are: $\{s_2 - s_3, s_2 - s_4, s_2 - s_5\}$.

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

of deep explanations as to why a certain item has been recommended or why no solution exists in a certain recommendation context [Fel+09]. 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*.

Knowledge-based recommendation technologies can support consistency management as well as intelligent explanations in situations where no release plan can be identified due to contradicting stakeholder preferences [Fel+09][Fel+10a]. Table 2.8 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 her/his preferences which can be 1 (*include*) or 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 Requirements Engineering ontology proposed by Lohmann et al. [LRA08]) but not used in this example. For the preferences shown in Table 2.4 no solution exists, that is, the stakeholder preferences are inconsistent (Tables 2.9 and 2.10).

Table 2.8: 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

Table 2.9: Example importance values for the stakeholder preferences shown in Table 2.8.

	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 2.10: 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

The first step to resolve this inconsistency is to figure out combinations of constraints (preferences) that are causes for the inconsistency, for example, the stakeholder preference c_{12} is inconsistent with the preference c_{22}. The complete set of such (minimal [Jun04]) 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 [Jun04]. We can now determine all possible repairs for the given set C of stakeholder preferences by simply deleting at least one element from each subset of CON (see [Rei87]). 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 [Fel+10a]) 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.

2.4 Issues for Future Research for Recommendation Technologies

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

Decision Support & Preference Construction. Existing Requirements Engineering approaches often 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 [FCM05][Fel+12]. 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 systems research is a new and challenging field of research which requires a strongly interdisciplinary research approach [Fel+12].

Recommendation Approaches. We exemplified how Requirements Engineering 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 [BFG11] 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 approaches have to be adapted for an optimal performance in Requirements Engineering 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 Recommender Systems. The position of stakeholders in a social network often has an enormous impact on Requirements Engineering 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 approaches can rely on concepts from the areas of natural language processing [FS05] and text mining [WF05].

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 Requirements Engineering 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 techniques for Requirements Engineering should be able to deal with graphical data sources such as, for example, class diagrams, sequence diagrams, state charts, as well as formal requirements specifications and system models. 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 Requirements Engineering related processes.

Context Awareness. There are two basic recommendation modes: *pull* and *push*. *Pull* means that stakeholders are actively triggering recommendation functionality when needed. *Push* means that the recommender application pro-actively detects situations (contexts) in which a stakeholder needs a particular support [HM08]. In order to deliver push recommendations, the context of a stakeholder has to be observed and discovered [HM08]. An approach to tackle this challenge is to continuously collect users' context and reason about user needs. Contextual recommendation is an emerging field [AM07] and will also play a major role in the development of recommendation solutions for Requirements Engineering.

2.5 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. In this chapter we focused on a discussion of existing research related to the application of recommendation technologies in different Requirements Engineering scenarios. In order to demonstrate the application of recommendation technologies we came up with examples such as the analysis of social networks for stakeholder identification and the clustering of requirements for the detection of dependencies. With our outlook on relevant topics for future research we hope to stimulate fruitful further research focused on the development and application of recommendation technologies in Requirements Engineering.

3 Human Factors in Requirements Engineering

This chapter is based on the work published in [Fel+12]. The contributions of the author of this thesis are the *study design*, an *in-depth literature analysis*, and the *writing of parts of the paper related user study*.

3.1 Introduction

Requirements Engineering is considered as one of the most critical phases in software projects [Poh96] and poorly implemented Requirements Engineering is a major risk for the failure of a project [HL01]. Requirements themselves are a verbalization of decision alternatives regarding the functionality and quality of the software [AW03]. Related individual as well as group decisions are extremely difficult due to the increasing size of requirement models as well as contradicting preferences of stakeholders [AP05][Cas+08].

In this chapter we analyze the impact of applying group recommendation technologies [Mas11][JBK04] to improve the quality of decision processes in the context of *requirements negotiation* which is the process of resolving existing conflicts between requirements and deciding which requirements should be implemented. Functionalities often provided by group recommenders are the visualization of the preferences of other group members, recommendations for individual and group decisions, and recommendations for conflict resolutions in the case of inconsistent stakeholder preferences [Mas11][JBK04]. Our motivation for applying group recommendation technologies is to improve the *usability* and the *quality of decision support* in Requirements Engineering environments (especially in the context of requirements negotiation).

Note that decision models based on rational thinking [McF99] are not applicable in most requirements negotiation scenarios since stakeholders do not exactly know their preferences beforehand [AP05][BJP98]. Furthermore, preferences are not stable but rather change over time which is an important aspect to be taken into account by requirements negotiation environments [AP05][BJP98]. The group recommendation technologies discussed in this

chapter are based on incremental preference elicitation [Bly02] and thus are key technologies for preference construction [PC08].

For the purpose of supporting preference construction in requirements negotiation we have developed the IntelliReq decision support environment. In our scenario, student teams are allowed to configure the set of requirements that should be implemented in their software project. Note that our goal was to develop recommendation technologies which can be flexibly exploited in requirements negotiation; it is not our intention to replace existing requirements negotiation approaches (see, e.g., [BGB01]) but to provide useful extensions.

The contribution of this chapter is the demonstration of the applicability of group recommendation technologies in requirements negotiation. We show that group recommendation technologies can be used to improve the perceived usability (in certain cases) and quality of decision support.

The remainder of this chapter is organized as follows. In Section 3.2 we introduce the IntelliReq environment which supports group decision processes for requirements negotiation. In Section 3.3 we present our hypotheses defined for the empirical evaluation of IntelliReq and discuss the corresponding study results. In Section 3.4 we discuss related work. The chapter is concluded with Section 3.5.

3.2 Requirements Engineering Environment

3.2.1 Application Scenario

For this study we used an early prototype of IntelliReq as group decision environment that supports computer science students at the Graz University of Technology in deciding on which requirements should be implemented within the scope of their software projects. Typically, a project team consists of 6–8 students who implement a software system with an average effort of about 8 man months. At the beginning of a project, students have to evaluate a set of requirements which have been defined by the course instructors and to figure out which requirements they will implement within the scope of their project (requirements negotiation phase). For example, the task could be the implementation of a tourist recommender application – the corresponding decision alternatives are depicted in Table 3.1. We will use this simple set of decision alternatives as a working example throughout the chapter.

Table 3.1: Example decisions to be taken by the project teams – taken decisions are interpreted as agreement between the project team and the course instructors. The fulfillment of the selected requirements is an evaluation criteria.

ID	Question	Decision Alternatives
1	which application domain?	20 destinations in Austria; world-wide
2	persistence management?	relational databases; XML Java objects
3	which type of user interface?	text-based; Java Swing Web application
4	recommendation algorithms?	knowledge-based collaborative & content-based
5	evaluation by whom?	students of own university other univ.; instructors
6	type of user manual?	HTML-based .pdf based
7	type of acceptance procedure?	live-demo slide presentation with screenshots

3.2.2 User Interface & Functionalities

Example screenshots of the IntelliReq user interface are depicted in Figures 3.1–3.3. With the goal of supporting the achievement of a common group decision, the IntelliReq user interface supports the following functionalities (the IntelliReq entry page is shown in Figure 3.1):

- Each stakeholder is enabled to define, adapt, and store her/his preferences (*add/change personal preferences*).
- Each stakeholder can comment on and discuss already defined preferences of other users (*show and comment on preferences of group members*).
- Each group can view and discuss recommendations for group decisions determined on the basis of already defined user preferences (*show group recommendation*).
- Define and store a group decision; this can only be done by the project manager (*edit current group decision*).
- Each IntelliReq user can evaluate the application (*evaluate* IntelliReq); this user feedback has been analyzed within the scope of an empirical study.

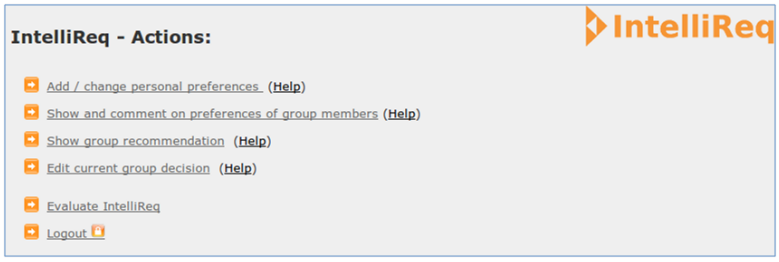

Figure 3.1: Activities supported by the INTELLIREQ user interface. Each group member can define and adapt his/her own preferences. These preferences can be seen and discussed by other group members. On the basis of articulated user preferences and a system-determined group recommendation, the team (represented by the project manager) can define and store the team (group) decision. Team decisions can be reviewed and adapted later on (until the submission deadline for team decisions has passed).

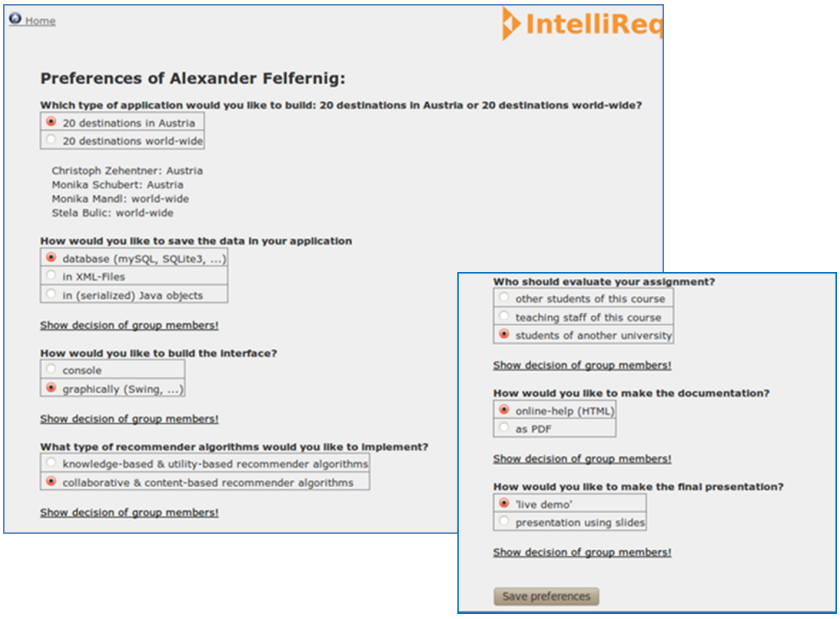

Figure 3.2: INTELLIREQ preference specification: each group member articulates his/her own preferences and – during this process – has insights into the preferences of other group members.

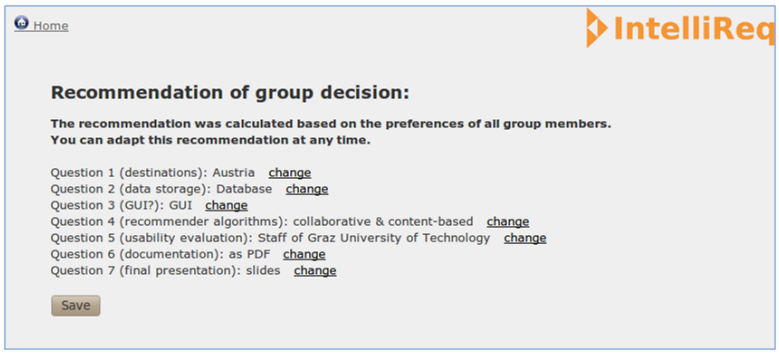

Figure 3.3: INTELLIREQ group recommendation.

3.3 Empirical Study

In order to evaluate the provided INTELLIREQ functionalities, we conducted an empirical study within the scope of the course *Object-oriented Analysis & Design* organized at the Graz University of Technology. The major focus of this study was to analyze the impact of group decision technologies on the dimensions *usability* of the system and *quality of decision support* in the context of Requirements Engineering.

3.3.1 Study Design

For the purpose of the empirical study we provided the INTELLIREQ environment in *four* versions. In order to analyze our hypotheses, we decided to implement a 2x2 study with the variation points *group recommendations available* (*yes*/*no*) and *preferences of other users visible* (*yes*/*no*) – these versions are shown in Table 3.2. Both, group recommendations and preference visibility, are key functionalities provided by state of the art group recommendation environments [Mas11][JBK04]. On the basis of this empirical study we wanted to investigate to which extent these functionalities are applicable within the scope of requirements negotiation.

Table 3.2: The four used INTELLIREQ versions. Variation points: *group recommendation supported* (*yes*/*no*) and *preferences of other team members are visible* (*yes*/*no*).

	with recommendation	without recommendation
preference view	version 1	version 3
no preference view	version 2	version 4

N=293 participants (computer science students at the Graz University of Technology, 23.1% female and 76.9% male) selected their preferred requirements using the INTELLIREQ environment. The participants were randomly assigned to one of 56 different groups (the

development teams) and defined (stored) 3733 individual preferences and 101 group decisions. For each development team the last stored group decision was interpreted as the final decision; after the published deadline no further adaptations of the taken decisions were possible. After a user had successfully articulated her/his requirements, she/he had the possibility to give feedback on the *usability* and the *decision support quality* of IntelliReq (*evaluate* IntelliReq link in Figure 3.1) on a 10-point Likert scale.

3.3.2 Study Hypotheses

The empirical study is based on hypotheses derived from existing research in the areas of Requirements Engineering [AP05][BGB01; NE00], group recommender systems [Mas11][JBK04][Fel+10b], and decision & social psychology [BJP98][Cia01][GS03][MS10]. The corresponding list of hypotheses is shown in Table 3.3.

Table 3.3: Hypotheses (H) for evaluating the group decision support environment.

H	description
H1	*group recommendations* improve the perceived system usability
H2	*group recommendations* improve the perceived quality of decision support
H3	*group recommendations* trigger more discussions
H4	*preference visibility* for all deteriorates the perceived usability
H5	*preference visibility* for all deteriorates perceived decision support quality
H6	*preference visibility* for all triggers less preference adaptations
H7	*preference visibility* triggers a decision bias
H8	winning strategy: use *group recommendation* but not support *preference visibility*
H9	*unconsidered preferences* deteriorate perceived usability & decision support quality

Group Recommendation (Hypotheses 1–3) Existing research in the field of recommender systems [Mas11][JBK04][Fel+10b] points out the potential of group recommendation technologies to significantly improve the quality of group decision processes. *First* we wanted

to investigate the potential of group recommendation technologies to improve the quality of the dimensions *usability* and *decision support* in a requirements negotiation scenario. With *Hypothesis 1* we express the assumption that recommendation technologies can improve the overall system quality in terms of *usability*. *Hypothesis 2* expresses the assumption that recommendation technologies can help to improve the perceived *quality of decision support*. *Second* we wanted to know whether the availability of group recommendations has an influence on the frequency of applying discussion functionalities (*Hypothesis 3*) – the underlying assumption is that the availability of group recommendations intensifies discussions between group members. This phenomenon is well known and exploited by critiquing-based recommenders where the system proposes recommendations and the user can give feedback in terms of critiques [PC08]. Studies in social psychology show that frequent information interchange can improve the decision quality [GS03][MS10].

Visible User Preferences (Hypotheses 4–7) Existing research in the field of group-based recommendation points out the advantages of preference transparency in group decision making [Mas11][JBK04]. In contrast, literature in social psychology points out the fact that suboptimal outcomes of group decision processes are correlated with the visibility of individual preferences of other group members [MS10][GS03]. The reason for groups not being able to take optimal decisions (hidden-profile identification problem) is explained by an insufficient exchange of decision-relevant information triggered by the initial disclosure of individual preferences (focus shift from information interchange to preference comparison). *First* we wanted to investigate whether the group-wide visibility of individual preferences has an influence on the perceived usability and decision support quality (*Hypotheses 4* and *5*). *Second* we wanted to figure out whether the group-wide visibility of individual preferences has an influence on the frequency of preference adaptation (*Hypothesis 6*). One underlying assumption here is that persons follow the phenomenon of *social proof* [Cia01], i.e., are doing or accepting things that others already did (accepted). The other underlying assumption is that persons tend to stick with their current decision due to the phenomenon of *consistency*[Cia01], i.e., the effect that published personal opinions are changed less often. *Third*, a lower frequency of information exchange can lead to a different decision outcome [GS03]. With *Hypothesis 7* we wanted to investigate whether the group-wide visibility of preferences can lead to a decision bias (the phenomenon of *social proof* [Cia01]).

Winning Strategy (Hypothesis 8) We wanted to provide an answer to the question which of the four different IntelliReq versions will be evaluated best regarding usability and quality of decision support. With *Hypothesis 8* we want to express the assumption that group recommendations improve the system usability as well as the decision support quality. In contrast, making preferences of other group members visible in the group decision

process deteriorates the system evaluation. Consequently, *version* 2 (see Table 3.2) should be evaluated best.

Distance Matters (Hypothesis 9) Finally, we wanted to provide an answer to the question whether the distance of a users's preference to the final group decision has an impact on the overall system evaluation. With *Hypothesis 9* we express the assumption that users with a low number of considered requirements will not be satisfied with the system usability and the decision support quality.

Group recommendation heuristics

The *majority* rule (applied in our empirical study) is a simple but very effective heuristic in group decision making [HK05]: each decision is taken conform to the majority of the votes of the team members. In addition to the majority rule, there exist a couple of further heuristics [Mas11] which can be applied when generating recommendations for groups, for example, the *fairness* heuristic which guarantees that none of the group members will be disadvantaged.[1]

3.3.3 Study Results

In order to identify statistically significant differences in the user quality feedback depending on the used IntelliReq version we conducted a series of two-sample t-tests. We will now discuss the results of our analysis.

Hypothesis H1 has to be rejected since the *usability* of IntelliReq versions with recommendation support is only better on the descriptive level (mean of 7.0 with vs. a mean of 6.42 without recommendation support) compared to versions without a recommendation support (see Table 3.4).

Hypothesis H2 can be confirmed since we could detect a significant better evaluation of the IntelliReq *decision support* for recommendation-enhanced versions ($p<0.001$) compared to versions without a recommendation support. Table 3.4 summarizes the results of this evaluation.

Hypothesis H3 can be confirmed as well since the number of comments on individual preferences is significantly higher in versions with provided group recommendations ($p<0.0015$) –

[1] Note that due to limited number of subjects (N=293) we were not able to compare the different recommendation heuristics with regard to the dimensions usability and quality of decision support. Such comparisons will be in the focus of future work.

Table 3.4: User feedback on recommendation support (mean, SD=std.dev.).

recommendation	usability	SD	decision support	SD
yes	7.0	1.67	7.07	2.03
no	6.42	2.47	5.21	2.96

see Table 3.5. Thus we can interpret group recommendations as a stimulating element for information interchange among group members which is a key factor for high-quality group decisions [MS10][GS03].

Table 3.5: Impact on information exchange frequency (SD=std.dev.).

recommendation	#comments (mean)	SD
yes	7.96	5.974
no	3.53	2.71

Hypotheses H4 and *H5* can not be confirmed since users with no access to the preferences of other group members did not provide a significantly better rating for usability and quality of decision support. However, on the descriptive level the evaluation of versions *without* preference visibility for all group members is better compared to versions *with* preference visibility (see Table 3.6).

Table 3.6: User feedback on IntelliReq preference accessibility (mean, SD=std.dev.).

preference access	usability	SD	decision support	SD
yes	6.46	2.09	6.16	2.72
no	7.0	2.08	6.25	2.64

Hypothesis H6 can be confirmed since the number of adapted individual preferences is significantly *lower* in versions with access to the personal preferences of other group members ($p<0.001$). This can be explained by the fact that – due to preferences visible for other users – the current user inclines to be *consistent* [Cia01] with his/her original requirements, i.e., the willingness to change articulated preferences decreases if preferences are accessible for other users [Cia01].

Hypothesis H7 can be confirmed since users having access to the preferences of other group members articulate preferences which are more similar to the final group decision (see Table 3.7). Being confronted with the preferences of other group members, persons base their decisions on the already known preferences and do not focus on the exchange of decision-relevant information which is extremely important for finding optimal decisions [GS03].

There is a significant biasing effect due to the visibility of preferences ($p<0.001$). This effect can be explained by the phenomenon of social proof [Cia01] which triggers group members to do things or accept things that other group members are doing (accepting).

Table 3.7: Impact of preference accessibility on user preferences (mean, SD=std.dev.).

preference access	distance of indiv. preferences	SD
yes	0.28	0.09
no	0.43	0.13

Hypothesis H8 can not be confirmed. However, users with recommendation support and without insight into the preferences of other users (INTELLIREQ version 2 – see Table 3.2) provided the highest ranking for both, *usability* and *quality of decision support* (see Table 3.8 and Table 3.9). Versions with recommendation support outperform versions without recommendation support in terms of *decision support quality* (see Tables 3.4 and 3.8) ($p<0.001$) and versions with recommendation support and without a view on the preferences of other users clearly outperform all other versions in terms of *usability* ($p<0.001$) – see Table 3.9.

Table 3.8: Impact of preference visibility and group recommendation on decision support quality (mean, SD=std.dev.).

version	recommendation	preference view	decision support quality	SD
1	yes	yes	7.03	2.04
2	yes	no	7.11	2.06
3	no	yes	5.13	3.07
4	no	no	5.29	2.91

Hypothesis H9 can be confirmed since users with preferences having a higher distance from the final group decision rated the INTELLIREQ environment significantly worse in terms of *usability* ($p<0.05$). This result conforms to the *win-lose* situations discussed in [BGB01] which typically turn into *lose-lose* situations. We could not detect a difference in the *quality of decision support* (see Table 3.10).

Table 3.9: Impact of preference visibility and group recommendation on usability (mean, SD=std.dev.).

version	recommendation	preference view	usability	SD
1	yes	yes	6.37	1.84
2	yes	no	7.62	1.21
3	no	yes	6.56	2.38
4	no	no	6.29	2.59

Table 3.10: Relationship between the distance of individual preferences to the final decision and perceived usability and decision support quality (mean, SD=std.dev.).

#answers different from group decision	usability	SD	decision support quality	SD
≤ 2 answers	7.05	2.04	6.18	2.72
> 2 answers	6.15	2.08	6.15	2.75

3.4 Related Work

Group recommender systems support human decision making by taking into account factors such as beliefs (knowledge) about the opinion of other group members, knowledge about individual motivations, and personal preferences. The major goal of group recommenders [Mas11][JBK04] is to achieve consensus among the members of the group – such a consensus is achieved by different heuristics such as *majority voting* (for each decision preferences with an underlying majority are selected) or *fairness* (a fair consideration of the preferences of each stakeholder).

In contrast to the results reported, for example in [JBK04], showing individual preferences to other group members is not always a good choice since this can lead to a lower perceived usability and decision support quality. This result is consistent with results of empirical studies conducted in the area of social psychology [MS10] where the outcome of group decisions significantly deteriorated when group members knew about the preferences of other group members. *Psychological studies* on the role of individual preferences in group decision making clearly show biasing effects in terms of significantly different outcomes of the decision process depending on whether preferences of other group members are known or not (see, e.g., [MS10]). This phenomenon can be explained by the fact that group members predominantly base their decisions on preferences known beforehand and not on the information generated in the decision process. As a consequence, optimal decisions (solutions) can only be identified if group members are not(!) confronted with individual preferences before starting a decision process. The failure of groups to identify acceptable or optimal decisions (so-called hidden-profile identification problem) can be explained by the insufficient discussion of unshared information (triggered, for example, by the articulation of initial preferences) and the resulting premature consensus of a group on an alternative which is not optimal [GS03].

Typical functionalities of group support systems in the Requirements Engineering context are brainstorming, idea organization, voting mechanisms, discussion forums, and shared drawing [BGB01]. Group support systems can help to significantly reduce Requirements Engineering efforts and achieve higher-quality results [BGB01]. Compared to integrated group support environments (see, e.g., [BGB01]), IntelliReq focuses on the specific aspect

of group recommendation. For existing requirements engineering environments (see, e.g., [BGB01]), the concepts presented in this chapter can contribute to achieve more effective decision processes. Note that IntelliReq technologies can be applied in the context of different negotiation constellations [HGR10] such as sales meetings, application requirements definition, reactive product line scoping, and release planning. Finally, we want to emphasize that models of human decision making on the basis of rational thinking [McF99] are not applicable in Requirements Engineering scenarios since preferences of stakeholders are not stable, i.e., change over time. Existing Requirements Engineering environments neglect this important aspect [AP05]. The group decision technologies presented in this chapter are based on incremental preference elicitation [Bly02] which provides a solid basis for handling unstable preferences [PC08].

The *application of recommendation technologies* in the context of Requirements Engineering is a constantly evolving research field [AP05][Cas+08]. The current research focus is on the application of machine learning approaches to the generation of coherent sets of requirements [Cas+08]. An example of the application of these technologies are users of open source CRM environments who perform badly when having to identify the appropriate discussion forum for a certain feature request. Another application of clustering techniques is introduced in [Che+05] where an intelligent requirement grouping mechanism is applied to support the construction of feature models. As far as we know there do not exist any applications of group recommendation technologies in the context of requirements negotiation. We see the results presented in this chapter as a first step to improve the overall decision quality in different phases of the Requirements Engineering process (e.g., evaluation, negotiation, and planning). For a comprehensive overview of potential application areas of recommendation technologies in requirements engineering we refer the reader to [MT09].

3.5 Conclusions

In this chapter we introduced an early prototype of the IntelliReq decision support environment which is used at the Graz University of Technology for supporting group decision processes in small-sized software projects (6–8 team members). Each group in the empirical study interacted with exactly one version of IntelliReq – the four versions provided differed in terms of the *availability of recommendation support* (yes/no) and the possibility to take a look at the *preferences of other users* (possible/not possible). The major results of this experiment were that group recommendation can improve the perceived usability (in specific cases) and quality of decision support. It is not recommended to disclose the preferences of individual group members at the beginning of a decision process since the knowledge of the preferences of other group members can lead to an insufficient exchange of

decision-relevant information. The results of our study clearly indicate that deep knowledge about human decision making can help to improve the overall quality of decision support environments. The investigation of further psychological issues is within the scope of our future research.

4 Recommending for Reuse and Dependency Detection

This chapter is based on the work published in [Nin+14a]. The contributions of the author of this thesis are the *algorithmic design of the proposed approaches*, the *design of the user study*, and the *writing of major parts of the paper*.

4.1 Introduction

Ensuring quality in a software project is a complex task. On the one hand there are limited resources in software projects. On the other hand there is a huge set of requirements which should be satisfied. Consequently, it must be guaranteed not to waste valuable resources on unnecessary tasks and to implement the most important artifacts first. A software development process can be divided into Requirements Engineering, architectural and detailed design, implementation, and testing. The scope of Requirements Engineering tasks is fairly broad as it starts with an unlimited solution space and is used to set borders for the following software development. It also needs to transform high-level objectives to operational prescriptions [Lam00][CA07].

According to a Gartner report [Gar11], corrections of defects (e.g., conflicting or missing requirements, wrong or incomplete artifact descriptions) are inexpensive during the Requirements Engineering phase but they are very expensive after delivery which makes decisions taken during the Requirements Engineering phase critical for the success of software projects [HL01]. Undetected errors in the Requirements Engineering phase is a major source of problems in subsequent development phases. These errors trigger not only costs for correcting the offending error, but also generate expenses in related artifacts to this error (e.g., redesign of code, documentation rewrite, and costs of the replacement of software already deployed) [LW00][GZ05]. Consequently, it is necessary to establish actions which guarantee requirements with high quality, address the stakeholders needs, and have no inconsistencies or errors [HRL12][NE00].

In most cases, to derive a complete definition of all needed objectives in a software development project, it is necessary to include a variety of different and inhomogeneous stakeholders in the Requirements Engineering process [EL02]. Within this group of participants it cannot be assumed that everybody has the expertise to write specifications in a formal representation. Thus, it is more attractive to evolve, maintain and discuss requirements in natural language with possibly non-technical customers as this is the only language which can be confidently assumed to be shared among all involved stakeholders in a software development process [Lam00][Cha06][Ger00][GN02].

Requirements analysts start with ill-defined and in many cases conflicting ideas of what the proposed system is expected to do, and must progress towards a single, detailed, technical specification of the system [CA07]. Within this process analysts are confronted with non-functional concerns such as safety, security, usability, performance, and so forth which are often conflicting with functional requirements [Lam00]. This fact and the circumstance that requirements are often presented without an explicitly specified structure complicates the Requirements Engineering process [GKB08]. Unfortunately, creating a structure in Requirements Engineering is a labor-intensive task as software projects consist of a huge amount of requirements. A complicating factor is that the process of defining a structure involves understanding of the needs of users, customers, and other stakeholders, and also of the context in which the to-be-developed software will be used [CA07]. Without the existence of formal descriptions, requirements validation usually describes a subjective evaluation of informal or undocumented requirements which requires stakeholders involvement [CA07].

As mentioned before, a subjective evaluation without any intelligent support is a labor-intensive and error prone task. Consequently, facilitating the task by preprocessing the provided data is promising. Previous empirical research explained that the identification of user requirements and the improvement of this task by automation is the most important activity [LMP04]. They also claimed the demand for a Computer Aided Requirements Engineering (CARE) tool based on Natural Language Processing (NLP). Additionally, it is concluded that linguistic techniques may play a crucial role in providing support for requirements analysis [LMP04]. One possible action is the automated processing of natural language requirements with a series of transformations such as tokenization, parts-of-speech tagging, parsing, and transformation into a set of logic formulas [GZ05].

Although the prospect of a support system that would automatically understand user's needs is very appealing, less sophisticated systems can sufficiently facilitate the work of requirements engineers [Rya93]. Tools can assist in various tasks such as scanning, searching, browsing and tagging requirement texts. For example, similarity analysis techniques give reasonably high accuracy considering its simplicity and can help to avoid assigning the same

requirements to different developers. Also for the release planning purpose and prioritization, interdependencies between requirements are necessary [Dag+02].

Beside the completeness of the Requirements Engineering specification and the conflict-freeness between requirements, the quality of the single artifacts is of high importance. Like in other engineering branches, the evolution of specifications over time and the partial reuse of already implemented specifications can be used to improve the artifacts description quality [NE00]. This is supported by two different benefits which go along with the evolution and reuse of artifacts: first, the identification of existing systems that could be transferred into the current software development project entirely or with a minimum of modifications. Achieving these goals reduces the effort of defining and developing new software products [CR00]. Second, knowledge about problems which can arise with a specific task can be identified and a solution can be provided. In any case, there is a good chance that the reuse of existing knowledge from previous work increases the quality and reduces the risk of failure in software projects. It is therefore a good advice to establish and maintain repositories for Requirements Engineering artifacts which share best practices [CA07].

Unfortunately, requirements reuse is often a complex task due to the fact that requirements are written in a natural language which circumvent the effective reuse. *To overcome this problem, requirements statements need to be processed in a unique fashion to accommodate reuse tasks, which include analysis of existing requirements, their organization into a repository of reusable requirement artifacts, and their synthesis into new requirements documents* [CR00]. Within this task several challenges arise like requirements belonging to each other without sharing the same words. For example, in a project description one requirement is about an interface for *clients* and another requirement describes the interface for *customers*. In this case these two requirements are related to each other. On the other hand there are requirements sharing common words without having a relation to each other. Therefore, *ambiguity* and *synonymy* are major problems in the context of requirements clustering [BT12].

The contributions of this chapter are two-fold: first, we propose two different recommendation algorithm implementations to support stakeholders in the Requirements Engineering process. Second, we conducted two studies at the Graz University of Technology to evaluate the applicability ot these techniques in the Requirements Engineering context. The remainder of this chapter is organized as follows. In Section 4.2 we summarize work related to the techniques used in this chapter. In Section 4.3 we present an overview of IntelliReq which is our web platform to develop and evaluate recommendation technologies. Section 4.4 describes the two used content-based recommender implemented for our evaluation. In Section 4.5 we present the findings of the conducted studies and in Section 4.6 we conclude

this chapter and outline future research directions.

4.2 Related Work

This section gives an overview about definitions and approaches related to text evaluation and similarity calculation.

4.2.1 Text Document Representation

A commonly used technique is the so called bag-of-words representation. Within this approach the information about paragraphs, sentences, and word orders are removed to make the information more useful for machine learning algorithms [SM99]. In this bag all non-descriptive words like *and* or *has* are defined as *stop words* and have to be removed. The remaining words are stemmed (reducing inflected or derived words to their stem) and their occurrence is stored in a vector [HSS03]. For example, if the word *house* and the word *houses* can be found in a text the stemmed version of both terms is *hous* and the resulting occurrence is two.

4.2.2 Word Sense

To find relations between requirements it is necessary to calculate the similarity of all words contained in the textual description of the involved requirements. It is stated that not the word form, but rather the *Word Sense* is the relevant participant to define a possible relation between words [AB06]. In WordNet, for example, these *Word Senses* are defined as *synsets* (short for synonymy sets) and can be interpreted as the ambiguity of the word [Voo93][MG10]. For example, the term *cold* has a different meaning in the sentence *a person is cold* as in the sentence *a room is cold*. To handle this ambiguity *synsets* can be used instead of terms within the bags-of-word representation. This leads to two benefits: first, the terms are fully disambiguated as the context has been taken into account and should increase precision. Second, equivalent terms can easily be identified as they all reside in the same *synset* which increases recall [Gon+98].

However, the assignment of the correct *Word Sense* to a term is a challenging task. It is necessary to decide whether to use a simple rule like taking the most frequent used *Word Sense* found in the used language or to analyze the complete context in which the word under investigation occurs. The second approach is clearly more complex as it needs to calculate the probability for all possible word senses of all terms used in a document [HSS03].

Next, the representation of the *Word Sense* inside the bag-of-words has to be chosen. There are basically three concepts [HSS03]:

- *Add Word Sense:* For each term add a *Word Sense*. This results in an occurrence of at least two, as the original term is not replaced.
- *Replace terms by Word Sense:* By replacing the original term the minimal occurrence can be one.
- *Word senses only:* The bag-of-words only contains an entry for terms where a *Word Sense* can be found. The *Word Sense* is used to replace the original term. The cardinality of this representation is smallest of the three presented options.

Alternatively to using already defined *Word Senses* like *synsets* one can discover *Word Senses* by clustering. With this approach similar *Word Senses* are derived from the context in which they are used. The assumption is that the meaning of an unknown word can often be inferred from its context. This approach is meant to cope with the problem that standard dictionaries miss domain-specific senses of words [PL02]. On the other hand, learning-based approaches are very domain-specific which means that the quality of the classifier drops strongly when the same classifier is used in a different domain [Tab+11].

4.2.3 Domain Knowledge

Domain knowledge is one crucial factor for high quality requirements elicitation [KS06]. A domain thesaurus can be used to formalize this knowledge and to inherit a classification of terms for further processing [CR00]. Initializing a domain thesaurus is labor-intensive in the factors cost and maintenance but also contain high-value knowledge. To reduce the initial effort of creating a domain thesaurus, *Wikipedia* can be used as a source of manually defined terms and relationships [MMW06].

4.3 IntelliReq

Having the need for computer aided software engineering (see Section 4.1) we decided to develop IntelliReq[1] which is a web platform for early Requirements Engineering. Besides the content-based recommendations discussed in this chapter, IntelliReq also supports group-based recommendations and stakeholder guidance techniques to improve the quality of the Requirements Engineering process [Nin+14b]. Figure 4.1 shows the latest GUI version of IntelliReq. We use IntelliReq to evaluate the applicability of Artificial Intelligence (AI) techniques for Requirements Engineering. IntelliReq also supports geographically

[1] http://www.intellireq.org

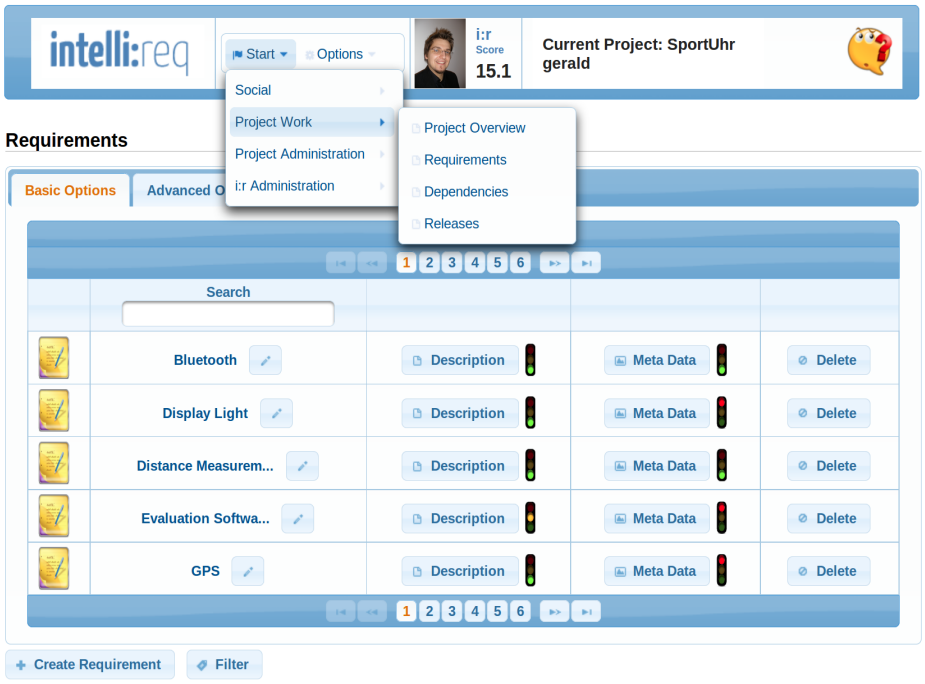

Figure 4.1: Requirements overview of the latest version of IntelliReq. Traffic lights are used for stakeholder guidance to improve the overall quality of artifacts. During our research the GUI evolved and has a different look than the version used for our evaluation of the *Keyword Recommender* (see Figure 4.2).

independent collaborative work which is often necessary when project stakeholders can not participate in meetings [Cas+09]. A main non-functional requirement for IntelliReq itself was the computational simplicity of all our recommender algorithms because our system is designed as an online multi-agent platform with fast response time.

Research has been done in clustering requirements to find dominant themes (topics) to support the assignment of potential interested stakeholders to this themes [Cas+09]. In our work we focus on the calculation of tensions between requirements, because we need a ranking of the k-top related requirements to a requirement under investigation. Although clustering has not been focused yet, the calculated tensions produced within IntelliReq can be used as input for a subsequent clustering.

In this chapter we discuss the content-based recommendation support evaluated with IntelliReq, which can mainly be divided into the following two techniques:

- *Keyword Recommender*: IntelliReq recommends keywords for a new inserted requirement

- *Dependency Recommender*: IntelliReq recommends requirement pairs as dependency candidates

The reason for these two recommender implementations were two-fold: first, we want to increase the reuse of artifacts which can, according to literature, increase the software quality [NE00]. This should be done with the *Keyword Recommender*. Second, we want to facilitate the management of requirements in the dimensions completeness, redundancy, and consistency. The scope of this recommendation is no automatic generation of dependencies but to recommend dependency candidates to the users for further investigation.

We discuss these two recommender implementations in more detail in Section 4.4 and the results of studies conducted at Graz University of Technology in Section 4.5.

Lexical Semantic Resources

To enhance the calculation of similarity with semantic information it is necessary to select a lexical semantic resource. We therefore discuss three different available resources and motivate our decision for our selection. Large lexical semantic resources can be categorized into *expert-built lexical semantic resources* (ELSR) like *GermaNet*[2] and *collaboratively constructed lexical semantic resources* (CLSR) like *Wiktionary*[3]. *OpenThesaurus*[4] can be located between these two definitions as it is collaboratively constructed, but reviewed and maintained by an administrator who revises all changes made in the database [MG10][Nab05].

All resources support *Hyponymy* which declares relationships between words as *Hyponyms* and *Hypernyms*. A *Hyponym* can be characterized as a type-of relationship while a Hypernym is a topic of a set of other terms. For example, we can denote the word *Measurement Device* as a *Hypernym* while the term chronometer is a *Hyponym* to *Measurement Device*.

OpenThesaurus follows the idea that users should be able to freely contribute to the project. The access to the stored data is available through their web portal, where users can search for synonyms. There is also an API for a web service and the data can be downloaded as a MySQL database dump file. The main focus of *OpenThesaurus* is to provide synonyms for words. This is based on two reasons: first, the project should be kept simple and second, the most prominent application is *OpenOffice* which has no strong demand for other relations than synonyms [Nab05].

Comparing these different lexical resources one can say that *GermaNet* contains the largest amount of taxonomic relations, *OpenThesaurus* provides the highest number of synonyms,

[2] http://www.sfs.uni-tuebingen.de/GermaNet/
[3] http://de.wiktionary.org
[4] http://www.openthesaurus.de

and *Wiktionary* contains the most antonyms and the second most synonyms and hypernymy relations [MG10].

While *OpenThesaurus* hardly supports *Hyponomy* it massively outperforms the other resources in the dimension *Synonym* relations (*OpenThesaurus*: 288,121, *GermaNet*: 69,097, *Wiktionary*: 62,235) [MG10]. For that reason we decided to use *OpenThesaurus*.

In *OpenThesaurus Word Senses* are grouped into *Synsets* (see Section 4.2). If we consider the example term *cold* then we get the *Synsets* with the numbers: 1110, 3834, 3945, 10632, 29416. Table 4.1 shows the corresponding terms for the *Synsets* 1110 and 3834.

Synset ID	Terms
1110	cold, insensible, cruel, icy, cold-hearted, hard-hearted
3834	cold, fresh, cool, frosty

Table 4.1: Synsets for the term *cold*.

Language Versions

We evaluated the *Keyword Recommender* within an empirical study conducted during the course Object-oriented Analysis and Design at Graz University of Technology. As not all course members had German language skills we used English as description language for requirements. On the opposite the second empirical study was conducted with German native speakers as we did not want to risk a bias based on lack of language skills when participants should evaluate the natural language processing capacity of IntelliReq. We therefore developed an English version for the evaluation of the *Keyword Recommender* and a German version for the evaluation of the *Dependency Recommender*.

4.4 Recommendation

4.4.1 Keyword Recommender

The *Keyword Recommender* is designed to support the English language and uses a simple generation for the bag-of-words. First, all stop words are removed from the text and the remaining words are transferred to a lower case representation which is stemmed using the *Porter Stemming Algorithm*[5]. The stemmed version and the original version are stored into a

[5] http://snowball.tartarus.org/algorithms/porter/stemmer.html

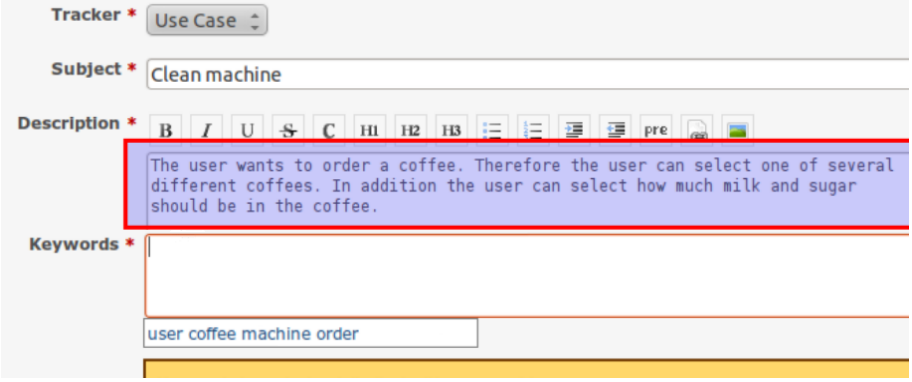

Figure 4.2: Screenshot of the *Keyword Recommender*.

lookup table.

To facilitate the reuse of requirements INTELLIREQ provides a *filtered by keyword* functionality. With this functionality users can select a keyword from a list to set the filter. Our keyword recommender stores all keywords used to annotate the requirements and the stemmed version of the keywords. Each time a new requirement is inserted, all words of the requirements subject and description are stemmed and compared to the internal list. If an already stored keyword is equal to one word of the new provided data, the recommender returns the originally used keyword for annotation (see Figure 4.2). With this approach we want to minimize the cardinality of words used to annotate the requirements in the database. For example, taken the term *database*. In the group without the keyword recommendation users use the keywords *database* and *databases*. Supported by the *Keyword Recommender* users get the keyword *database* as replacement for *databases* proposed. Figure 4.2 shows the requirements creation window with the keyword recommender.

We also used a knowledge thesaurus to enhance the quality of the recommendation. This thesaurus was manually created to support the definition of software applications in the domain of recommender technologies for tourism. We therefore defined word synonyms for a controlled subset of terms and declared only one word sense for each term. For example, we defined the words: *tourist*, *client*, *customer*, *holidaymaker*, *vacationer* as synonym and these words did not occur in any other synonym list. Using this domain-specific recommender it is possible to increase the efficiency of the *Keyword Recommender* like it is proposed in literature (see Section 4.2).

The time should be displayed {}{9135, 11112, 11532}{17111} {1331, 3527, 10772, 11615, 20552}{}

Table 4.2: Fragment text of a requirement. The last two lines show the bag-of-words where empty braces represent removed words

4.4.2 Dependency Recommender

For our *Dependency Recommender* we start the generation of the bag-of-word similar to the implementation of the *Keyword Recommender*. First, the text is stripped of all stop words and transferred to the lower case representation. Next, the tokens need to be converted in a comparable form. Instead of using a stemmer, the *Dependency Recommender* uses a mapping table to get the base forms of the terms. For this purpose we take the *Morphy*[6] data file which consists of a list of German terms with all inflected forms and grammatical properties and generated a SQL table with the data. With this table a query for the German plural *Häuser* (houses) will result in the singular 'Haus' (house). The *Morphy* data set contains 368,175 associations between inflected and base forms. Furthermore, these base forms facilitate the lookup in the *OpenThesaurus* database because the original dataset contains no stemmed forms. Next, we store the data inside the bag-of-words in the *Word Senses only* representation (see Section 4.2), which means that we only consider words which matches an entry in the *OpenThesaurus* database.

We also need to state definitions for the involved elements. We define a document (requirement) in a project as $d \in D$ with D is the set of all documents of a project. Words of a document are defined in the set W_d, S_w is defined as a set of *Word Senses*, and $w \in W_d$ is a word in the document. The cardinality[7] of the associated *Word Sense* set to this word w is defined as $|S_w|$. Taking the term *time* from Table 4.2 the word *time* is $w \in W_d$, the values 9135, 11112, 11532 are in the set of *Word Senses* S_w and the cardinality of $|S_W|$ is 3.

With this definition we generate the bag-of-words by replacing all words in the document by a list of *Word Senses*. If a word can not be found in the *OpenThesaurus* database it will not be considered. Table 4.2 shows a text fragment of a requirement. If a word was identified as stop word or did not find a match in *OpenThesaurus* it was removed. To keep our *Dependency Recommender* computationally fast we define some simplifications. First, as we do not have enough information of *Hypernyms* and *Hyponyms* in the OpenThesaurus data

[6] http://www.danielnaber.de/morphologie/

[7] Note that this cardinality is document independent as these sets are defined in *OpenThesaurus*.

set these relations are ignored. Second, we assume that within a project two identical terms have the same *Word Sense*. We assume that this fact does not hold in any case, but as this approach works for the creation of the knowledge thesaurus for our *Keyword Recommender*, we suppose that the amount of terms with different *Word Senses* within a project domain is small enough to not heavily deteriorate the quality of the *Dependency Recommender*.

Based on these simplifications we define the tension between two terms *w1* and *w2* in Formula 4.1. In this Formula the value $|S_{w1}|$ is the cardinality of term *w1* and $|S_{w2}|$ is the cardinality of term *w2*. Also, the value *Matches* defines how many *Word Senses* these two terms have in common.

$$tension(w1,w2) = \frac{Matches}{|S_{w1}|} + \frac{Matches}{|S_{w2}|} \tag{4.1}$$

To clarify Formula 4.1 we discuss the following three situations:

- (a) Strong match: There are a lot of equal *Word Senses* between the two terms under investigation. This will lead to a high impact on the calculation of the similarity.
- (b) Weak match: Only a few *Word Senses* are equal between the two terms under investigation. The association between these two terms will only have a small impact on the calculation of the similarity.
- (c) No match: None of the *Word Senses* of the two terms are equal. The association between these two terms will have no impact on the calculation of the similarity.

For example, we take a look at the two terms *cruel* and *cool* from Table 4.1. We can calculate a cardinality of 6 for the term *cruel* and a cardinality of 4 for the term *cool*. As there is only one matching *Synset* we can calculate the tension as $\frac{1}{6} + \frac{1}{4} = \frac{5}{12}$ which is very low. Of course, two equal terms will result in the highest possible tension with the value of 2 as the *Matches* are equal to the cardinality.

To further enhance the recommendation of candidates for dependencies between two requirements we exploit the knowledge about the topics of requirements. This knowledge is explicitly provided by stakeholders as they define the requirement names. To clarify this approach we discuss the example shown in Figure 4.3. Both requirements *Height Determination* and *Speed Measurement* mention the third requirement *Internal Memory* in their description. As all three requirements use the two terms *Internal* and *Memory*, the similarity calculation would recommend them as equal similar to each other. Our assumption is that there is a higher tension between terms used in one requirement as topic than between terms only used in the description. We therefore *doubled* the value for *Word Senses* used in requirements topics for our similarity calculation.

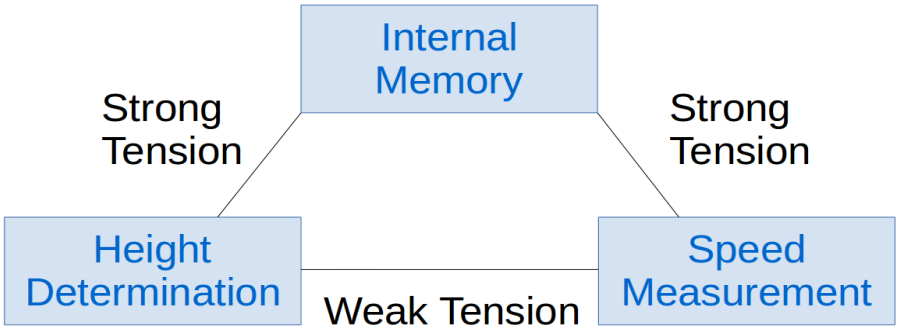

Figure 4.3: Shows the tension calculation for the term *Memory* between the three requirements *Internal Memory*, *Height Determination*, and *Speed Measurement*. All three requirements descriptions also contain the term *Memory* with an occurrence of one.

4.4.3 Reduce Dimension

In a next step we want to reduce the dimension of keywords used for the recommendation. We therefore use *Apache OpenNLP*[8] which is a toolkit for natural language processing. With this toolkit we filter out all terms which are not classified as noun. The remaining terms are then used for the calculation of the similarity and can be presented as explanation for the recommendation.

4.4.4 Calculating Requirements Similarity

For the calculation of the similarity between two requirements in a project three well known measurements can be used: the *Dice*, *Jaccard*, and *Cosine* coefficients. The *Dice* coefficient can be found in Formula 4.2, which is a variation of the Jaccard coefficient "intensively" taking into account keyword commonalities [Dag+02][Jan+10].

$$sim(r_a, r_b) = \frac{2 * |Keywords_A \bigcap Keywords_B|}{|Keywords_A| + |Keywords_B|} \tag{4.2}$$

Instead of user defined keywords we used the calculated tensions from Formula 4.1 to enhance the calculation with the knowledge derived from *OpenThesaurus*.

[8] https://opennlp.apache.org

4.5 Empirical Studies

This section covers the results of the studies conducted at our University with the recommender proposed in Section 4.4.

4.5.1 Keyword Recommender

Within the scope of a study conducted at the Graz University of Technology we evaluated the quality of the aforementioned recommendation approaches. The empirical study has been conducted within the course Object-oriented Analysis and Design (N=39 software teams of size 5-6; 15.45% female, 84.55% male). In this context the teams had to create 20 requirements for a software project. The creation of requirements was defined as an collaborative task and the students were encouraged to reuse[9] requirements already defined by other groups.

To evaluate the effectiveness of the *Keyword Recommender* we randomly assigned the development teams to two groups. The first group had no recommender, while the second group was supported by the *Keyword Recommender* described in Section 4.4. Both groups used an own database to store requirements. For example, if a team of the first group generated a requirement all teams from the first group could access this requirement and were able reuse it. Teams from the second group could not see or reuse requirements from the first group and vice versa.

All different study groups had an interface to browse through the requirements presented as an unsorted list. Additionally, teams could filter the input by selecting keywords.

Evaluation

When evaluating the reuse behavior between the two study groups we could identify a significant increase (t-test, $p < 0.05$) of reuse activity throughout the teams in the study group with *Keyword Recommender*. Table 4.3 shows the results of our evaluation. From this result we derived that teams with a keyword recommendation used more often the same keywords to annotate requirements. For example, two different requirements were annotated by the keyword *database* instead of *database* and *databases*[10]. Also using an example from

[9] Note that a reuse is not a link to a requirement. Instead the requirement is cloned and the reusing team links to the new version of the requirement. Therefore, any changes done to an reused requirement did not affected the original requirement and vice versa.

[10] The annotation with *database* and *databases* could be found in the study group without the *Keyword Recommender*.

Group	Keyword Recommender	Reuse Requirement	Distribution
1	No	141	39.83%
2	Yes	213	60.17%
Total	-	354	100%

Table 4.3: Number of reused requirements in the two study groups.

our initial domain-specific thesaurus, users did not need to search requirements for *tourist* and *client* separately. As these words were characterized as synonyms in the domain used for the evaluation, the *Keyword Recommender* proposed the keyword *tourist* each time the term *client* or *tourist* was contained in a requirement description. This resulted in a shorter filter list of keywords which was used to facilitate the search through the list of reusable requirements. For example, teams with this advantage did not need to evaluate requirements found with *tourist* and *databases*. They retrieved all related requirements with a single click. One drawback of this approach was that it always uses the first written keyword. For example, if there is a typo in the written keyword (e.g. databasO), this incorrect diction will be used as a recommendation for similar keywords with the same stemmed version. Of course, using a thesaurus and a correction with the Levenshtein distance could reduce this problem, but we used a very simple approach without any typo correction.

4.5.2 Dependency Recommender

To evaluate the dependency detection approach we created a set with 30 requirements for a sport watch during a brainstorming session. The requirements covered functionality such as internal memory, training evaluation, connectivity to a PC system, and sensor measurements like heart rate. The evaluation of potential dependencies between the requirements was not covered in the brainstorming.

In a next step we applied our *Dependency Recommender* on the set of requirements to generate a ranked list of potential dependencies. According to Formula 4.3 with $n = 30$ (number of requirements in our project) there exist 435 possible dependencies between two requirements. We also discovered a significant decrease of the calculated similarity value after the top 20 recommendations. To evaluate the quality of the calculated recommendation we conducted a second study at Graz University of Technology (N=23 participants; 8.69% female, 91.31% male). As we were only interested in evaluating the quality of the recommendation we printed out the recommendations and presented them to study participants offline. We want to point out that the *Dependency Recommender* is integrated in the online

Name	REQ_0002
Topic	GPS
Description	To collect position data a GPS system should be used. Using this data in combination with speed measurement enables the calculation of the covered distance .
Keywords	GPS
Category	GPS

Figure 4.4: Shows the requirement representation used in our study.

version of IntelliReq. All recommendations are calculated on demand and there is no necessity for precedent offline calculations[11].

During the study the participants had to decide if the recommended requirement pairs are *worth a look* for the task of finding dependencies. Our approach was not designed to automatically find dependencies, but to support stakeholders with a preselection of interesting dependency candidates for evaluation. Finally, the participants were asked to rate the overall usefulness of the recommendation on a 5-point scale with 1 (very useful) to 5 (not useful). The results can be found in Figure 4.5. Based on the decrease of the calculated similarity value we presented only the best 20 recommendations as we assumed a high increase in false-positive recommendations based on the low calculated similarity values [Dag+02]. This should prevent a bias of the overall satisfaction which would occur by showing many recommendations with very low calculated similarity.

Figure 4.6 shows the result of the acceptance evaluation for the first 20 recommendations. The value *agree* can be seen as *true-positive*, while *disagree* counts for *false-positive*. The dimensions *true-negative* and *false-negative* were not covered as for this the participants would have to evaluate all 435 possible dependency candidates to find pairs for good recommendation not already presented by the *Dependency Recommender*.

$$allPossibleRec = n * (n-1)/2 \tag{4.3}$$

To evaluate the study result we define a recommendation as accepted if more than 75% of the participants agreed on their usefulness (see Formula 4.4). Using this Formula we see that 70% of the first 10 recommendations were accepted. Although the quality of the

[11] Note that in the current version only the German language is supported for the similarity calculation.

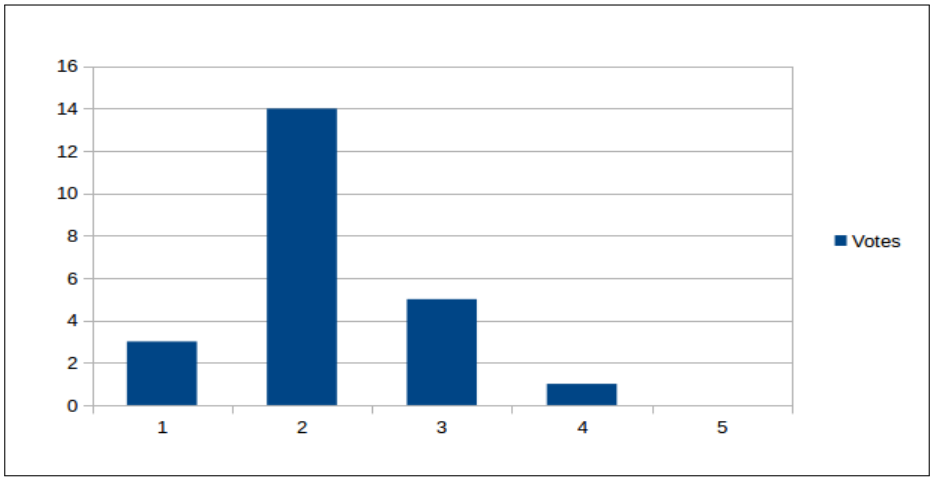

Figure 4.5: Shows the quality rating of study participants (1-Best, 5-Worst).

recommendations deteriorates for the next 10 recommendations we can still notice that 60% of the recommendations were supported.

$$accepted(agreed) = \begin{cases} agreed > 75\% & true \\ else & false \end{cases} \tag{4.4}$$

4.6 Conclusion

Existing Requirement Engineering tools primarily support the definition and cataloging of requirements but fail to provide additional information such as similarity of requirements. Although there has been a lot of research done to fully understand text written in natural language it has been pointed out that semi-automated approaches provide a good way to balance human skills and computer tools [BI96]. We therefore concentrated on the support of stakeholders defining dependencies and / or reusing requirements instead of trying to find a fully automated approach.

We evaluated our recommendation techniques in IntelliReq which are calculated online without any necessary offline precomputation. This was an important criteria as our platform is defined as multi-agent system where different stakeholders can contribute to the Requirements Engineering process at the same time.

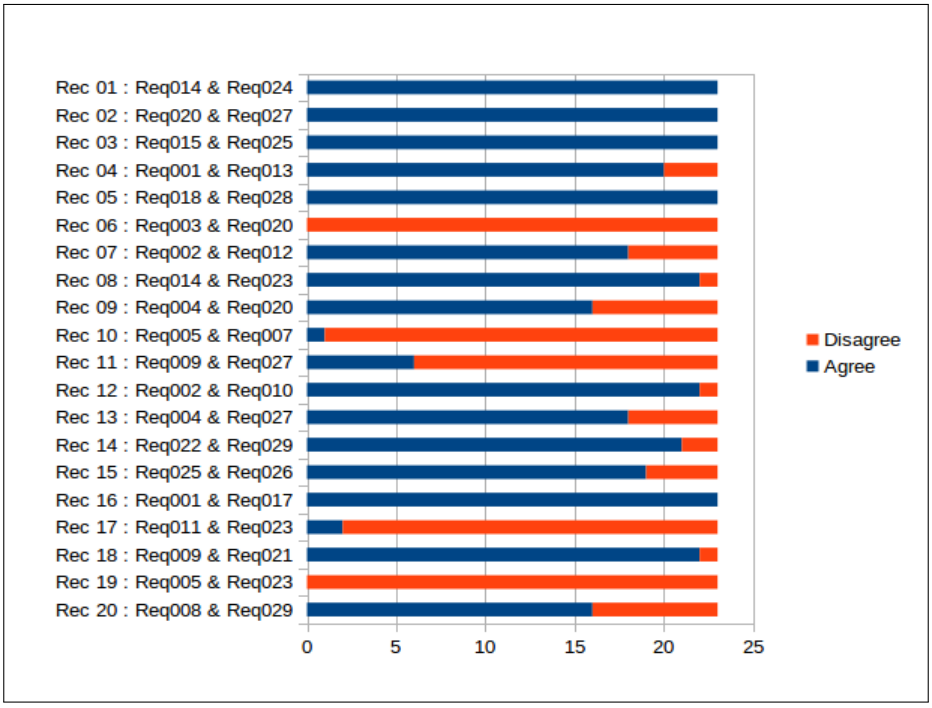

Figure 4.6: Shows the relation between true-positive (users agrees with recommendation) and true-negative (user disagree with the usefulness of the recommendation). User feedback was collected for the 20 recommendations with the highest similarity score.

We enhanced our *Keyword Recommender* with domain knowledge in the form of a manually generated thesaurus and defined one *Word Sense* for each term. During evaluation of this approach we discovered a significant increase of reuse activity by software development teams using our *Keyword Recommender*. Furthermore, we conducted a study to evaluate the quality of our similarity measurement technique used in the *Dependency Recommender*. We took 20 recommendations from 435 possible combinations between requirements with the highest calculated similarity score and presented them to study participants. Although our proposed approach is rather trivial, 13 of 20 recommendations were accepted by study participants. Also, the perceived usefulness of this kind of recommendation was rated high with an average of 2.17.

Quality of Dependency Detection. In order to further improve the quality of dependency detection mechanisms in IntelliReq, approaches from natural language processing [FS05] and text mining [WF05] have to be combined with content-based approaches currently included in IntelliReq. We also want to evaluate the applicability of micro-tasks within groups of stakeholders (as used, for example, in *Amazon Mechanical Turk*) to generate and maintain domain-knowledge thesauri to enhance our content-based recommendation techniques [IPW10].

5 Recommending Prioritizations

This chapter is based on the work published in [FN12], [FNR12], [NFR12], and [Nin12]. The major contributions of the author of this thesis are an *in-depth literature analysis*, the *algorithmic design and development of the presented new heuristics*, the *design of the user study*, and the *writing of major parts of the papers*.

5.1 Introduction

Within software projects there are several fundamental facts [Fir04]:

- *Difference in importance:* not all proposed requirements are equally important
- *Limited project resources*: usually it's impossible to implement all requirements
- *Long schedule*: systems development requires many months or years, during which requirements are subject to significant changes
- *Small Requirements Engineering budget*: available budget for Requirements Engineering tasks rarely exceeds 2-4% of the project budget

Prioritizations support software project managers in the systematic definition of subsequent software releases. Especially due to the resource limitations in software projects it is essential to use a systematic prioritization process to implement the most important requirements in software projects first [HL01][Wie99]. Typically, requirements prioritization is a collaborative task as developers cannot always estimate the most important requirements to a customer and customers cannot evaluate the cost and technical difficulties of specific requirements [Wie99]. However, establishing consensus between stakeholders regarding the prioritization of a given set of requirements is a challenging task. On the other hand, properly prioritizing leads to significant benefits such as *improved customer satisfaction*, *lower risk of cancellation*, *force stakeholder to address all requirements and not just their own*, and *prioritize investments* (e.g., allocate limited resources for quality assurance only for the most important requirements) [Fir04].

Requirements prioritization is a specific type of group work which becomes increasingly important in organizations [PH97]. Requirements Engineering processes such as EasyWinWin

[BGB01] (which is based on the risk-oriented spiral model [Boe88]) include prioritization operations which are based on the assumption that stakeholders know their preferences. An example technique to the implementation of requirements prioritization assuming such stable stakeholder preferences is based on the concepts of *quality function deployment* (QFD) [Wie99]. This approach provides a structured way of identifying a prioritization for requirements. All the requirements are enlisted in a table and for each requirement the responsible stakeholders have to define the benefit (business value of the requirements in case it is implemented) and the penalty of not implementing the requirement. Both together form the relative value of the requirement. Beside the value of a requirement we have to estimate the corresponding development risks, for example, the non-availability of the needed expertise and resources and corresponding technological risks. The priority of a specific requirement r can then be determined on the basis of multi-attribute utility theory [WE86] with the *interest dimensions* d_1: *value* and d_2: *risk* (see Formula 5.1) where $w(d_i)$ denotes the weight of an interest dimension d_i. The result of applying Formula 5.1 is a recommendation for the prioritization of a given set of requirements.

$$Priority(r) = \sum_{d=0}^{n} w(d_i) * d_i \tag{5.1}$$

Such utility-based approaches rely on the basic assumption of preference stability which means that stakeholders exactly know their preferences, i.e., are able to evaluate requirements with regard to the interest dimensions d_i. In contrast to this assumption, most real-world decisions are based on preference construction [BJP91], i.e., stakeholders are developing their preferences within the scope of a decision making process. Requirements Engineering processes not taking into account this fact are running the risk of low-quality requirements prioritizations due to the fact that the human decision processes are not taken into account [AW03]. A mechanism to improve the applicability of requirements negotiation tools is to proactively assist stakeholders in their personal decision making process [AW03][FN12]. In Chapter 3 we showed that group recommendation technologies [Mas11] can significantly improve the usability and decision support quality of Requirements Engineering environments. Furthermore, group recommendation can stimulate information exchange in requirements negotiation scenarios which results in group decisions of higher quality [GS03]. Studies in social psychology clearly show that frequent information exchange (of decision relevant information) between group members can significantly improve the quality of a group decision [GS03]. The effect of increased information exchange can be explained by the fact that the availability of group recommendations intensifies discussions between group members. The phenomenon is well known and exploited, for example, by critiquing-based recommender systems [CP12] where the system proposes recommendations and users can provide feedback in terms of critiques. In a group recommendation scenario, such critiques are defined in terms

of preference adaptations which are in the following the input for the calculation of new group recommendations. Note that we interpret group recommendation as basic mechanism to support decision processes. However, decisions are still taken by engaged stakeholders and group recommendation should help to improve the overall decision quality and the efficiency of decision making.

Another challenge is the primacy effect in preference elicitation [Fel+07]: the outcome of this phase will depend on the sequence in which preferences have been inserted. The psychological literature shows that consensus about topics formed early in discussions is cognitive resistant to changes. Additional information added later will be assimilated to already chosen consensus and it is very unlikely that another option is chosen [LKT01]. This phenomenon can be explained by the *assimilating effect* based on the dissonance theory [Fes57] which states *that individuals are motivated to reduce psychological incongruity or discrepancy* that may arise by adding new information to a present perception [CNR04]. The result is that stakeholders will perceive already selected options more attractive than new options [CNR04] and this leads to a bias of group preferences depending on the order of the incoming preferences. Unfortunately, this effect is increased if there is a high group identity, because the fear of exclusion from the group is higher [LKT01]. To reduce this effect, a brainstorming phase in which stakeholders become aware of their own preferences should be established. To raise the willingness of stakeholders to report their honest concerns, this brainstorming phase should be implemented in an anonymous fashion [Grü00]. In a software environment, for example, this can be done if requirements can be added without authentication. First related study results will be presented in the following sections.

An important issue in the group prioritization process is the factor of *fairness*. The degree of perceived fairness influences the willingness of group members to accept compromises in the resolution of disputes and their trust in other stakeholders [LKT01]. Especially in environments with a high amount of requirements it is necessary to provide tool support in the prioritization process to achieve a maximum degree of perceived fairness. One way to provide such a tool support is to present recommendations of reasonable requirements prioritizations to decision makers.

In this chapter we investigate different *recommendation heuristics* with respect to the achieved *prediction quality*. We also show how anonymity (preferences are not connected to stakeholder names) in group decision processes can help to improve the quality of requirements prioritizations. The major contributions reported in this chapter are to show that anonymity in decision making influences the quality of the prioritization and to present new heuristics for group recommendations with better prediction quality.

The remainder of this chapter is organized as follows. In Section 5.2 we provide an overview

of the IntelliReq decision support environment developed at the Graz University of Technology. In Section 5.3 we show the impact of the factors *anonymity*, *consensus*, and *decision diversity* on the *quality of the prioritization*, the *stakeholder satisfaction* with the prioritization, and the *quality* of software artifacts. Section 5.4 covers standard recommendation heuristics including our new proposed heuristics. In Section 5.5 we conclude this chapter with an outlook on future work.

5.2 IntelliReq Decision Support

Based on the findings presented in Chapter 3 we improved the IntelliReq system which is a Requirements Engineering environment. IntelliReq was used to support computer science students at our university in deciding on which requirements should be implemented within the scope of their software projects.

For this task 219 students enrolled in a course on *Object-Oriented Analysis and Design* at the Graz University of Technology had to form groups of 5–6 members. Unfortunately, it is not possible to evaluate the knowledge and experience of the students and the resulting groups but the course is typically attended by students in the third semester of an computer science program or similar. We therefore assigned the resulting groups randomly to different evaluation pools and assumed that the knowledge and experience is equally distributed on each pool.

In our study, 39 software development teams had to define a set of requirements for a software system with an average effort of about 8 man months which in the following had to be implemented. These requirements had to be prioritized and the resulting prioritization served as a major criteria for evaluating the delivered software product at the end of the project.

The requirements prioritization process consisted of three different phases (see Figure 5.1) denoted as *construction* (collection of individual stakeholder preferences), *consensus* (discussion of prioritization alternatives and adaptation of own preferences), and *decision* (group decision defined and explained by the project manager). This decision process structure results in about 10.000 stakeholder decisions and 798 corresponding group decisions (39 groups with approximately 20 final decisions per group) taken by the team leaders (project managers).

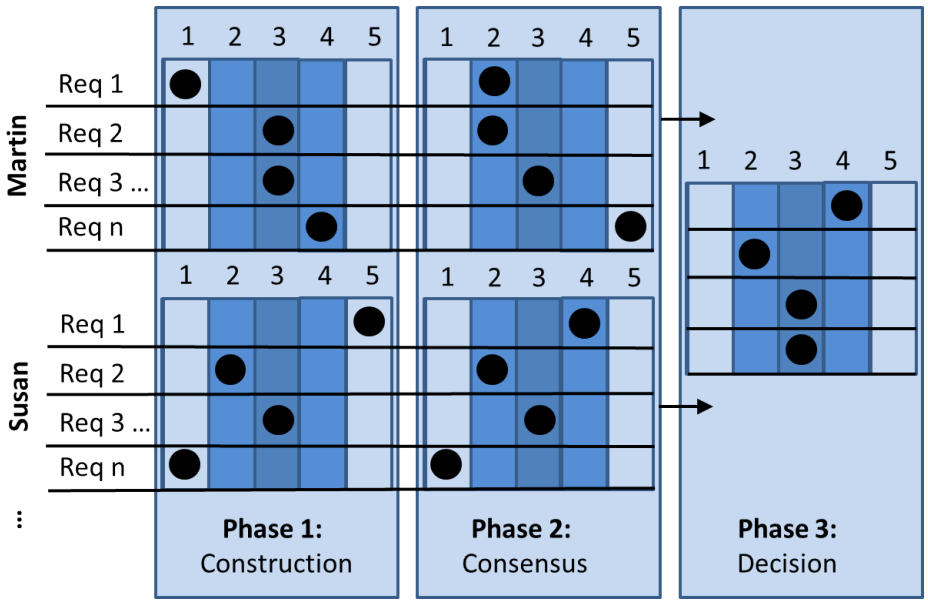

Figure 5.1: IntelliReq Prioritization (Decision) Process. *Construction*: stakeholders define their initial preferences; *Consensus*: stakeholders adapt their preferences on the basis of the knowledge about preferences of other stakeholders. *Decision*: project managers take the final group decision. Preferences represent the wish of a stakeholder to implement a requirement (1: lowest, 5: highest)

5.3 Anonymous Preference Elicitation

5.3.1 Motivation

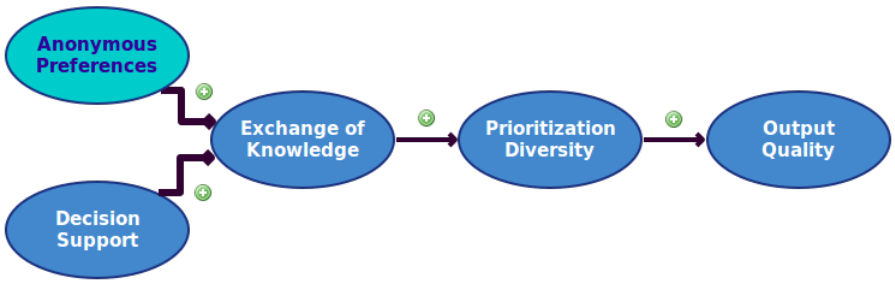

Figure 5.2: The impact of *Anonymous Preferences* in IntelliReq on the final *Output Quality*

Prioritization decisions are typically taken in groups but this task is still ineffective due to reasons such as social blocking, censorship, and hidden agendas [PH97]. The major contribution of this section is to show how *anonymity* in group decision processes can help to improve the quality of requirements prioritizations. We therefore assume that an *anonymous preference elicitation* has an positive impact on the *output quality* (see Figure 5.2). Further-

more, anonymous preference elicitation increases the probability of detecting hidden profiles [GS03], i.e., increases the probability of exchanging decision-relevant information [MS10].

Group recommender systems support human decision making by taking into account factors such as beliefs (knowledge) about the opinion of other group members, knowledge about individual motivations, and personal preferences. The major goal of group recommenders [Mas11][JBK04] is to achieve consensus among the members of the group – such a consensus is achieved by different heuristics such *majority voting* (for each decision preferences with an underlying majority are selected) or *fairness* (a fair consideration of the preferences of each stakeholder).

In contrast to the results reported, for example in [JBK04], showing individual preferences to other group members is not always a good idea since this can lead to a lower perceived usability and decision support quality. This result is consistent with results of empirical studies conducted in the area of social psychology [MS10] where the outcome of group decisions significantly deteriorated when group members knew about the preferences of other group members. *Psychological studies* on the role of individual preferences in group decision making clearly show biasing effects in terms of significantly different outcomes of the decision process depending on whether preferences of other group members are known or not (see, e.g., [MS10]). This phenomenon can be explained by the fact that group members predominantly base their decisions on preferences known beforehand and not on the information generated in the decision process. As a consequence, optimal decisions (solutions) can only be identified if group members are not(!) confronted with individual preferences before starting a decision process. The failure of groups to identify acceptable or optimal decisions (so-called hidden-profile identification problem) can be explained by the insufficient discussion of unshared information (triggered, for example, by the articulation of initial preferences) and the resulting premature consensus of a group on an alternative which is not optimal [GS03].

Typical functionalities of group support systems in the Requirements Engineering context are brainstorming, idea organization, voting mechanisms, discussion forums, and shared drawing [BGB01]. Group support systems can help to significantly reduce Requirements Engineering costs and achieve higher-quality results [BGB01]. Compared to integrated group support environments (see, e.g., [BGB01]), IntelliReq focuses on the specific aspect of group recommendation. For existing requirements engineering environments (see, e.g., [BGB01]), the concepts presented in this chapter can contribute to achieve more effective decision processes. Note that IntelliReq technologies can be applied in the context of different negotiation constellations [HGR10] such as sales meetings, application requirements definition, reactive product line scoping, and release planning. Finally, we want to emphasize

that models of human decision making on the basis of rational thinking [McF99] are not applicable in Requirements Engineering scenarios since preferences of stakeholders are not stable, i.e., change over time. Existing Requirements Engineering environments neglect this important aspect [AP05]. The group decision technologies presented in this chapter are based on incremental preference elicitation [Bly02] which provides a solid basis for handling unstable preferences [PC08].

The *application of recommendation technologies* in the context of Requirements Engineering is a constantly evolving research field [AP05][Cas+08]. The current research focus is on the application of machine learning approaches to the generation of coherent sets of requirements [Cas+08]. An example of the application of these technologies are users of open source CRM environments who perform badly when having to identify the appropriate discussion forum for a certain feature request. Another application of clustering techniques is introduced in [Che+05] where an intelligent requirement grouping mechanism is applied to support the construction of feature models. As far as we know there do not exist any applications of group recommendation technologies in the context of requirements negotiation. We see the results presented in this chapter as a first step to improve the overall decision quality in different phases of the Requirements Engineering process (e.g., evaluation, negotiation, and planning). For a comprehensive overview of potential application areas of recommendation technologies in Requirements Engineering we refer the reader to [MT09].

5.3.2 Empirical Study

Within the scope of our empirical study we wanted to investigate the impact of *anonymous preference elicitation* on the decision support quality of the IntelliReq environment. Consequently, each project team interacted with exactly one of two existing types of preference elicitation interfaces. One interface (*type 1: non-anonymous preference elicitation*) provided an overview of the personal preferences of team members where each team member was represented by her/his name. In the second type of interface (*type 2: anonymous preference elicitation*) the preferences of team members were shown in anonymized form where the names of the individual team members were substituted with the terms $person_1$, $person_2$, *etc.* The hypotheses (H1–H8) used to evaluate the decision process are summarized in Figure 5.3. These hypotheses were evaluated on the basis of the following observation variables.

Anonymous preference elicitation. This variable indicates with which type of prioritization interface the team members were confronted (either summarization of the preferences of the team members including the name of the team members or not including the name of the team members).

Consensus and Dissent. An indication to which extent the team members managed to achieve consensus (dissent) – see the second phase of the group decision process in Figure 5.1 – is provided by the corresponding variables. We measured the *consensus of a group* on the basis of the standard deviation derived from requirement-specific group decisions. Formula 5.2 can be used to determine the *dissent* of a group x which is defined in terms of the normalized sum of the standard deviations (sd) of the requirement-specific votings. The group *consensus* can then be interpreted as the counterpart of dissent (see Formula 5.3).

$$dissent(x) = \frac{\sum_{r \in Requirements} sd(r)}{|Requirements|} \tag{5.2}$$

$$consensus(x) = \frac{1}{dissent(x)} \tag{5.3}$$

Decision Diversity. The decision diversity of a group can be defined in terms of the average over the decision diversity of individual users in the consensus phase (see Figure 1). The latter is defined in terms of the standard deviation derived from the decision d_u of a user – a decision consists of the individual requirements prioritizations of the user.

$$diversity(x) = \frac{\sum_{u \in Users} sd(d_u)}{|Users|} \tag{5.4}$$

Output Quality. The output quality of the software projects conducted within the scope of our empirical study has been derived from criteria such as degree of fulfillment of the specified requirements. We also weighted the requirements according to their defined priority in the prioritization task. E.g., not including a very high important requirement enormously decreases the quality value. Consequently, defining a high priority for a requirement with minor (or no) importance is a prioritization failure as this requirement has to be implemented or, otherwise, to risk an enormously reduction of the quality value. On the opposite, low priority requirements will only have a small impact on the quality value (independent from the real importance of the requirement for the project). Therefore, requirements prioritization has a direct impact on the quality value which has been graded by teaching assistants who did not know to what type of preference elicitation interface (anonymous vs. non-anonymous) the group has been assigned to. These assignments were randomized over all teaching assistants, i.e., each teaching assistant had to evaluate (on a scale of 0..100 credits) groups who interacted with an anonymous and a non-anonymous interface.

Within the scope of our study we wanted to evaluate the following hypotheses.

H1: Anonymous Preference Elicitation increases Consensus. The idea behind this hypothesis is that anonymous preference elicitation helps to decrease the commitment [Cia01] related

to an individual decision taken in the preference construction phase (see Figure 5.1), i.e., changing her/his mind is easier with an anonymous preference elicitation interface. Furthermore, anonymous preference elicitation increases the probability of detecting hidden profiles [GS03], i.e., increases the probability of exchanging decision-relevant information between stakeholders [MS10].

H2: Anonymous Preference Elicitation decreases Dissent. Following the idea of hypothesis H1, non-anonymous preference elicitation increases commitment with regard to already taken (and published) decisions. It also decreases the probability of detecting hidden profiles [MS10] and thus also decreases the probability of high-quality decisions.

H3: Consensus increases Decision Diversity. As a direct consequence of an increased exchange of decision-relevant information (see Hypothesis H1), deep insights into major properties of the decision problem can be expected. Also, having users with decreased commitment to their preferences about requirements enables decision making in a more analytical way (users are more willing to change their preferences). For example, having a majority of users with low preferences for a single requirement and one user with a high preference, finding a consensus about the priority for this requirement is easier if the commitment of the opposing user is lower. Of course, this should not exclude any discussion about the requirement itself to discover tacit knowledge. As a consequence, the important differentiation between important, less important, and unimportant requirements with respect to the next release [Dav03] can be achieved.

H4: Dissent decreases Decision Diversity. From less exchange of decision-relevant information we can expect a lower amount of globally available decision-relevant information. Also, the willingness of users to change their preferences to find a consensus is lower and necessary compromises, if based on users commitment instead for objective reasoning, reduces the quality of the decision making process. As a consequence, the differentiation between important, less important, and unimportant requirements is a bigger challenge for the engaged stakeholders.

H5: Consensus increases Output Quality. From Hypothesis H3 we assume a positive correlation between the degree of consensus and the diversity of the group decision. The diversity is an indicator for a meaningful triage [Dav03] between important, less important, and unimportant requirements.

H6: Dissent decreases Output Quality. In contrary, dissent leads to a lower decision diversity and – as a consequence – to less meaningful results of requirements triage.

H7: Decision Diversity increases Output Quality. Group decision diversity is assumed to be a direct indicator for the quality of the group decision. With this hypothesis we want to

analyze the direct interrelationship between prioritization diversity and the quality of the resulting software.

H8: Anonymous Preference Elicitation increases Output Quality. Finally, we want to explicitly analyze whether there exists a relationship between the type of preference elicitation and the corresponding output quality.

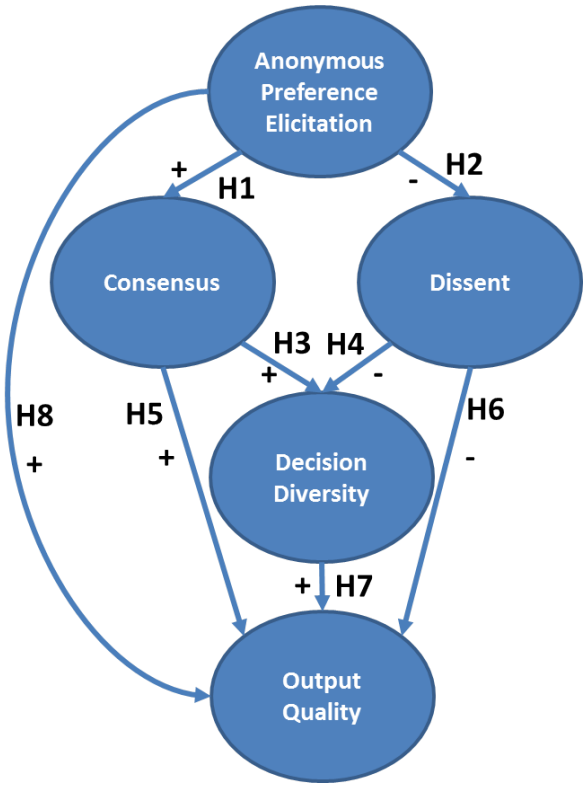

Figure 5.3: Hypotheses defined to evaluate the IntelliReq Decision Support.

5.3.3 Study Results

We analyzed the hypotheses presented above (H1–H8) on the basis of the variables introduced in Section 5.3.2.[1]

H1. The degree of group consensus in teams with anonymous preference elicitation is significantly higher compared to teams with non-anonymous preference elicitation (Mann-Whitney U-test, $p < 0.05$). An explanation model can be the reduction of commitment

[1] We are aware of the fact that *dissent* is the inverse function of *consensus*, however, for reasons of understandability we decided to explicitly include *dissent* as a decision variable.

[Cia01] and a higher probability of discovering hidden profile information which improves the overall knowledge level of the team.

H2. Group dissent is an inverse function of group consensus and – as a consequence – teams with non-anonymous preference elicitation have a significantly higher dissent (Mann-Whitney U-test, $p < 0.05$). In this context, non-anonymous preference elicitation can lead to higher commitment with regard to the originally articulated preferences.

H3. There is a positive correlation between the group consensus and the corresponding decision diversity (correlation 0.523, $p < 0.01$). More group discussions can lead to a higher level of relevant knowledge about the decision problem distributed among stakeholders. In the following this can lead to a development of a deeper understanding of the need of requirements triage [Dav03]. Additionally, decreased commitment can increase the willingness to accept compromises (based on objective reasoning) which makes the important differentiation between important, less important, and unimportant requirement easier. Deeper understanding of the need of requirements triage and a more objective reasoning lead to a higher degree of decision diversity.

H4. Dissent is an inverse function of group consensus – the higher the dissent, the lower the corresponding decision diversity (correlation -0.523, $p < 0.01$). A lower degree of group decision diversity (prioritization diversity) can be explained by a lower degree of decision-relevant knowledge and a lower willingness to make compromises.

H5. Consensus in group decision making increases the output quality (correlation 0.399, $p < 0.01$). An overlap in the personal stakeholder preferences can be interpreted as an indicator of a common understanding of the underlying set of requirements. This leads to a better prioritization and a higher quality of the resulting software components.

H6. The hypothesis can be confirmed (correlation -0.399, $p < 0.01$), i.e., there is a negative correlation between group dissent and the corresponding output quality.

H7. In our analysis we could detect a positive correlation between group decision diversity (diversity of prioritization) and the corresponding output quality (correlation 0.311, $p < 0.01$). Decision diversity can be seen as an indicator of a reasonable triage process and reasonable prioritizations result in higher-quality software components.

H8. Groups with anonymous preference elicitation performed significantly better compared to groups with a non-anonymous preference elicitation (independent two-sample t-test, $p < 0.05$).

5.4 Group Decision Support

Group recommendation technologies have been successfully applied in different scenarios where groups of people are in the need of decision support [Mas11]. For example, a group of people is interested in spending the holidays together or celebrating the birthday of a person in a restaurant. In the first scenario, the group has to take a decision regarding the holiday destination, in the second case the group has to develop a consensus regarding the restaurant for the birthday celebration.

This section covers the applicability of group recommendation approaches to requirements prioritization. In the following paragraphs we exemplify basic group recommendation heuristics which are applied in a requirements prioritization scenario. In this simplified example we assume that the requirements r_1, r_2, and r_3 have to be prioritized by the stakeholders Martin, Susan, Peter, and Pauline. Each of the given requirements can be prioritized on a scale 1 (unimportant) and 5 (very important). In our example we assume that the stakeholders have already specified their initial preferences and that the group recommender system aggregates the initial user preferences (prioritizations) into a recommendation for a group decision. In the following paragraphs we introduce the basic group recommendation heuristics [Mas11] least distance, majority voting, average value, and random priority selection. In Section 5.4.3 we analyze the predictive quality (precision) of the heuristics on the basis of a real-world dataset collected in software projects at the Graz University of Technology. Note that these heuristics currently do not take into account importance weights of specific stakeholder votes. For example, if a stakeholder has a very important reason as to why a requirement should be included in a certain release, this information is not taken into account by the discussed decision heuristics. In such a case we assume that the other stakeholders are convinced by the one stakeholder to change their mind. However, the inclusion of the importance of individual votes is within the scope of our future work.

5.4.1 Group Decision Heuristics

We hypothesize that the effects described in the previous section could be enhanced by the use of group recommendation heuristics. Consequently, we want to evaluate the best heuristics to predict stakeholder voting behavior in the requirements prioritization process. In a study conducted at our university we collected a dataset of the preferences of stakeholder and the resulting prioritizations of the software teams. With this dataset we evaluated different group recommendation heuristics [Mas11] and proposed three new *Group decision heuristics*.

and compared predicted with real decisions taken by the participants of our study. For the evaluation we used the precision metric according to Formula 5.5. The following subsections describe the different applied heuristics that are based on following parameters: **median** m**, standard deviation** sd**, heuristic** h and **requirement** r.

$$precision(h) = \frac{\#\ correctly\ predicted\ group\ preferences(h)}{\#\ predicted\ group\ preferences(h)} \tag{5.5}$$

Least Distance

This heuristic determines (recommends) for each requirement r_i the priority value p with the lowest distance to the other elements in the set of distinct user preferences (PREF) where $p \in PREF$. This criteria is formalized in a corresponding evaluation function and can be found in Formula 5.6.

$$priority(r) = selectmin(p, \sum_{pref \in PREF} |pref - p|) \tag{5.6}$$

A corresponding example is shown in Table 5.1. In this example, the recommended priority for requirement r_1 is 2, since 2 has the lowest distance to all other user preferences (prioritizations) $pref \in PREF$ = 1,2,3.

	Martin	Susan	Peter	Pauline	recom. priority
r_1	1	3	1	**2**	**2**
r_2	5	**4**	**4**	3	**4**
r_3	**2**	1	3	1	**2**

Table 5.1: Application of *Least Distance* heuristic.

Majority Voting

This heuristic recommends a priority value which represents the majority of stakeholder votes related to a specific requirement. An example for the recommendation result of the majority heuristic is given in Table 5.2.

	Martin	Susan	Peter	Pauline	recom. priority
r_1	**1**	3	**1**	2	**1**
r_2	5	**4**	**4**	3	**4**
r_3	2	**1**	3	**1**	**1**

Table 5.2: Application of *Majority Voting* heuristic.

Average Value

This heuristic determines the average value (see Formula 5.7) and round the result (see Formula 5.8) of the declared stakeholder preferences for each requirement. This value is then taken as a recommendation of the priority value for the corresponding requirement. An example for the application of the average value heuristic is shown in Table 5.3.

$$AVG(r) = \frac{\sum_{i=1}^{\#user} pref(i,r)}{\#user} \tag{5.7}$$

$$Priority(r) = \begin{cases} rounddown(AVG(r)) & AVG(r) < 0.5 \\ roundup(AVG(r)) & AVG(r) \geq 0.5 \end{cases} \tag{5.8}$$

Table 5.3: Application of "Average Value Heuristic"

	Martin	Susan	Peter	Pauline	recommended priority
r_1	1	3	1	2	**2**
r_2	5	4	4	3	**4**
r_3	2	1	3	1	**2**

Random Priority Selection

The random heuristic has been integrated only for evaluation purposes. This heuristic has - as expected - the weakest performance, which can be seen in Section 5.4.3. Table 5.4 shows the application of the *Random Priority Selection*. It can be observed that there is no functional relation between the votings and the recommendation.

Table 5.4: Application of "Random Priority Selection"

	Martin	Susan	Peter	Pauline	recommended priority
r_1	1	3	1	2	**4**
r_2	5	4	4	3	**1**
r_3	2	1	3	1	**2**

5.4.2 Advanced Group Decision Heuristics

Median Based

Inspired by a survey of the voting behavior of six-person juries conducted by the University of Chicago [SSK00], we implemented a new heuristic called *Median Based Heuristic* (see Formula 5.9). In the original study [SSK00], jury members were asked about their initial preferences for the penalty. These initial punishment preferences were compared with the final decision after group deliberating. The finding was that *there was a severity shift for the high-punishment cases, and a leniency shift for the low-punishment cases* [SSK00]. Such choice shifts are explained by the *Group Polarization Theory* [ZCW92].

In a similar fashion, our algorithm calculates the recommendations depending on the median of the initial preferences. In our study, the preferences are distributed on a six point scale. In this context, the algorithm has three possible states (see also Formula 5.9):

- The median is one or two: Calculate the average (see Formula 5.7) of the preferences and round **down** the result
- The median is three or four: Use the *Least Distance* heuristic (see Formula 5.6).
- The median is five or six: Calculate the average (see Formula 5.7) of the preferences and round **up** the result

$$Priority(r,m) = \begin{cases} rounddown(AVG(r)) & m = 1,2 \\ LDIS(r) & m = 3,4 \\ roundup(AVG(r)) & m = 5,6 \end{cases} \tag{5.9}$$

Ensemble Based

For a further improvement of the prediction quality we introduced a combination of different heuristics. For each recommendation task the algorithm calculates the *Majority* (MAJ), the *Least Distance* (LDIS), and the *Median Based* (MDB) heuristic. If two heuristics recommend the same result, this result is the final recommendation. If there are three different

recommendations, the *Median Based* heuristic is the final recommendation. The heuristic is shown in Formula 5.10.

$$Priority(r,m) = \begin{cases} MDB(r,m) & MDB(r,m) = MAJ(r) \\ MDB(r,m) & MDB(m,r) = LDIS(r) \\ LDIS(r) & LDIS(r) = MAJ(r) \\ MDB(r,m) & otherwise \end{cases} \tag{5.10}$$

Standard Deviation Based

The hypothesis for this heuristic is that the best heuristic depends on the degree of conformity of the initial preferences. For example, if the group has a high degree of consensus, the *Least Distance* heuristic will perform best. If there is a high dissent, the *Average* heuristic will perform better. The highest standard deviation in our dataset is 2.50. We divided the dataset into three subsets. Each subset has the same standard deviation range ($\frac{2.50}{3} \approx 0.84$). Next we tested the heuristics mentioned in this chapter on the dataset to find the heuristic which performs best on each subset (see Table 5.5). For this evaluation we used the complete dataset of the *Consensus* and *Decision* phases (see Figure 5.1) which included the anonymous and the non-anonymous setting. The combination of the different heuristics can be found in Formula 5.11.

$$Priority(sd,r,m) = \begin{cases} LDIS(r) & sd < 0.84 \\ MDB(r,m) & 0.84 \geq sd < 1.67 \\ AVG(r) & 1.67 \geq sd \leq 2.50 \end{cases} \tag{5.11}$$

Group	SD From	SD To	Best Heuristic
1	0	0.84	LDIS
2	0.85	1.67	MDB
3	1.68	2.50	AVG

Table 5.5: Groups based on *Standard Deviation*.

Thereafter we generated a function which uses the heuristic depending on the standard deviation and the results of Table 5.5.

5.4.3 Empirical Study

An overview of the study results can be found in Table 5.6.[2] Although the *Median Based* heuristic is outperformed by the *Least Distance* (see Table 5.6), the *Median Based* performs better in an environment with visible preferences. In the study conducted at the University of Chicago [SSK00], on which this heuristic is based on, the same observation was made in decision making processes where group member preferences are visible. The *Standard Deviation* heuristic is out of competition as this heuristic is defined for this specific dataset. Future studies with new datasets will show whether this combination of different heuristics will have an improved prediction quality. When comparing the prediction quality of the remaining heuristics, the *Ensemble* heuristic performs best.

Heuristic	Consensus	Decision
LDIS (least distance)	0.619	0.733
MAJ (majority voting)	0.576	0.719
AVG (average value)	0.617	0.702
RAN (random selection)	0.167	0.188
MDB (median based)	0.618	0.732
ESB (ensemble)	0.629	**0.739**
SDB (sd based)	**0.636**	0.722

Table 5.6: Comparison of heuristics of the *Consensus* and the *Decision* phase (best results are marked bold).

5.5 Conclusion

Requirements prioritization is an important task in software development processes. In this chapter we motivated the application of requirements prioritization and discussed issues related to the aspect of anonymizing group decision processes in requirements prioritization. The results of our empirical study clearly show the advantages of applying anonymized preference elicitation, for example, in terms of higher-quality software components, and can be seen as a step towards a more in-depth integration of decision-oriented research in the requirements engineering process.

The heuristics discussed in this chapter are basic group recommendation heuristics [Mas11]. Our initial analysis shows the applicability of these heuristics in terms of prediction quality. As part of our future work we want to investigate the positive effect of recommendations

[2] Due to the limited space only the results without differentiation between visible and non-visible preferences are shown.

on the dimensions *decision diversity*, *satisfaction with requirements prioritization*, and the *software quality* resulting from the requirements prioritization process. In addition, we want to conduct an in-depth evaluation of the user acceptance of the determined group recommendations - up to now only the majority voting based recommendations has been analyzed [Fel+12]. Furthermore, we want to analyze if the *Standard Deviation* heuristic (as described in Formula 5.11) has the same prediction quality on other data sets .

We also want to use this heuristic for group recommendation in our upcoming INTELLIREQ user studies. An important task for future work is the analysis of preference reversal [CNR04] in Requirements Engineering and the impact on the prioritization quality and the stakeholders' satisfaction with the prioritization. In this context we want to develop new methods to improve stakeholder satisfaction.

Another topic of interest is the different interpretation of preferences. Are the group members arguing of the same topic? Does everybody has the same understanding of a given term? It has been shown that team members often enter a decision process from different viewpoints. Therefore it is necessary to find a consensus on the interpretation of shared information. This is especially important as the interpretation of issues has an massive impact on the decision making and is therefore considered as crucial [MR01].

6 IntelliReq Prototype

This chapter is based on the work published in [Nin+14b]. The author of this thesis contributed the *related work*, the *development of the* IntelliReq *environment*, the *design of the user study*, and *wrote major parts of the paper*.

6.1 Introduction

Requirements Engineering 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 software's environment, and assumptions about the environment* – see [Dav03]. Major phases of a Requirements Engineering process are elicitation & definition, quality assurance, negotiation, and release planning [Som11]. Requirements Engineering is a critical phase of a software development project since low-quality requirements are a major reason for the failure of a project [HL01]. The corresponding follow-up costs can add up to 40% of the overall project costs [Lef97].

Due to the increasing size and complexity of software systems, there is a growing demand for intelligent approaches that can help to improve the quality of Requirements Engineering processes [Fel+10b][MT09][MC11][Ren+13]. Existing Requirements Engineering tools primarily support the definition and cataloging of requirements but fail to provide additional information such as hidden relationships between requirements and quality status of requirements. Furthermore, these tools do not support decision making scenarios where mediation support is needed (e.g., in the case of contradicting opinions and preferences of stakeholders). Finally, no mechanisms are integrated that help to increase user involvement although a low degree of involvement in many cases leads to project failure [LQF10].

In this chapter we introduce the IntelliReq environment[1] which exploits *recommendation technologies* [BFG11][Jan+10] to support Requirements Engineering tasks. IntelliReq supports *Early Requirements Engineering* where the major focus is to figure out and prioritize high-level requirements in software projects. The outcome of IntelliReq is a consistent

[1] http://www.intellireq.org

Table 6.1: Example of a content-based filtering recommendation scenario (REQ = set of requirements).

$r_i \in REQ$	category	release	effort	description
r_1	database	1	140 hours	store portfolio configuration in database
r_2	UI	1	300 hours	configurator UI with online help available
r_3	database	1	100 hours	Hibernate based database access
r_4	UI	2	200 hours	configurator UI with corporate identity

set of high-level requirements with corresponding effort estimations and a release plan for the implementation of the identified requirements. All information units (e.g., requirements, dependencies, and release plan) are summarized in a high-level specification book. IntelliReq is applied by the industry and research partners of the Graz University of Technology.

The remainder of this chapter is organized as follows. In Section 6.2 we show how recommender systems can support the development of requirements models. In Section 6.3 we present the IntelliReq user interface and discuss related functionalities. An overview of empirical studies related to the IntelliReq environment and the business benefits is given in Section 6.4. Section 6.5 contains a discussion of related and future work. The chapter is concluded with Section 6.6.

6.2 IntelliReq Recommendation Technologies

In this section we provide an overview of recommendation technologies integrated in IntelliReq. 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* [Bur00][BFG11]. Recommender systems support the identification of relevant items in situations where the complexity of an item assortment outstrips a user's capability to survey it and to reach a decision [Bur02].

6.2.1 Recommendation Approaches

There are the following *four basic types* of recommendation approaches.

Collaborative Filtering [Her+04][LSY03] is an implementation of word-of-mouth promotion where purchase decisions are taken on the basis of the opinion of relatives and friends: if users *A* and *B* rated similar items in a similar fashion in the past, collaborative filtering will propose new items to user *A* that *B* already rated positively.

Content-based Filtering [PB97] exploits features (e.g., keywords) of items a user liked in the past for the determination of recommendations. For example, if a customer of amazon.com bought books related to the Java programming language, similar books (related to Java) will be recommended in the future.

More complex items such as financial services or apartments are recommended by *knowledge-based recommenders* [Bur00][FB08]. In this case, constraints define the relationship between user requirements and the corresponding items and are thus responsible for the determination of recommendations.

Finally, *group recommenders* [Fel+12][JBK04][Mas11] recommend items for groups of users (e.g., recommendation of a hotel to a group of tourists who plan a common holiday trip).

6.2.2 Recommendation Approaches in IntelliReq

The following discussions are based on a simplified scenario which includes the four requirements depicted in Table 6.1. On the basis of this scenario we show how recommendation approaches can be exploited to support different types of Requirements Engineering activities (e.g., requirements definition, quality assurance, and release planning). In this context we want to emphasize that recommenders are key-supportive technologies, however, we do not claim that information gaps in general can be tackled by their application. For example, efficient Requirements Engineering processes heavily rely on the personal communication between stakeholders which cannot be substituted by recommenders. Based on our scenario we now exemplify the application of recommendation approaches in Requirements Engineering.

Content-based Filtering. Content-based Filtering [PB97] exploits similarities between user preferences and descriptions of items (items not known to the user up to now). User preferences are often represented in terms of keywords extracted from textual item descriptions – see also [MR00]. Alternatively, items can be described in terms of categories (semantic

Table 6.2: Keywords extracted from the textual requirement descriptions in Table 6.2 (*REQ* = set of requirements).

$r_i \in REQ$	extracted keywords
r_1	store, portfolio, configuration, database
r_2	configurator, UI, help
r_3	database, Hibernate
r_4	configurator, UI

descriptions). Typical recommendations determined by content-based filtering are of type *item A is recommended since you purchased item B which is similar to A*.

When defining requirements, stakeholders can be supported, for example, by pointing out requirements defined by other stakeholders in the current project that are similar to the current one. Furthermore, requirements can be recommended for reuse, i.e., requirements already defined in previous projects can be more easily retrieved and reused in the current project. The similarity between two requirements in the set *REQ* of defined requirements ($\{r_a, r_b\} \subseteq REQ$) can be determined on the basis of Formula 6.1 (Dice coefficient which is a variation of the Jaccard coefficient "intensively" taking into account keyword commonalities – see also [Jan+10]).

$$sim(r_a, r_b) = \frac{2 * |keywords(r_a) \cap keywords(r_b)|}{|keywords(r_a)| + |keywords(r_b)|} \tag{6.1}$$

For example, sim(r_1, r_3) = 0.33, since *keywords*(r_1) = {*store, portfolio, configuration, database*} and *keywords*(r_3) = {*database, Hibernate*} (see Table 6.2). Let us assume that the active stakeholder (*st*) has already investigated the requirement r_1. Now, content-based filtering would recommend requirement r_3 if r_3 has not been investigated by the active stakeholder up to now. If available, metadata can as well be exploited for determining the similarity between requirements – in this situation, keywords (see Formula 6.1) have to be substituted by category descriptions (see Table 6.2).

In IntelliReq, dependency detection is based on content-based filtering. Dependency detection is the task of identifying semantic relationships between requirements. Examples of relationships between two requirements (r_a and r_b) are r_a requires r_b, r_a is incompatible with r_b, r_a refines r_b, and r_a is part of r_b. IntelliReq does not identify relationships between requirements on a semantic level but on the level of similarities, i.e., the basic assumption is that similarity between requirements can be an indication of dependency. The assertion of a concrete dependency is the task of the stakeholders. In addition to the afore discussed content-based filtering approach, IntelliReq exploits semantic information extracted from

Table 6.3: Example of a decision problem: deciding about the group evaluation of requirement r_1 (MAJ = majority voting as decision heuristic).

r_1	st_a	st_b	st_c	st_d	**MAJ**
quality	medium	medium	medium	high	**medium**
priority	high	high	medium	high	**high**
decision	accept	revision	accept	accept	**accept**

OpenThesaurus[2] instead of keywords. This OpenThesaurus enhanced version is integrated in the current IntelliReq version (http://www.intellireq.org).[3]

Group Recommendation. The major goal of group recommendation technologies [Fel+12] [JBK04][Mas11] is to foster consensus among group members. Group recommenders can support group decision making by taking into account the fact that individual decisions depend on factors such as own evaluation of an alternative, beliefs about group member opinions, and information about the individual motivations (e.g., egocentric or cooperative motivation [JBK04]). Group recommenders include heuristics [Mas11] that can be used for identifying alternatives that will be accepted by all or at least a majority.

Requirements evaluation & negotiation are basic application scenarios for group recommenders since stakeholders have to cooperatively decide about the quality of requirements and also to figure out in which way requirements should be taken into account in the release plan. For demonstration purposes we assume that the requirement r_1 has been evaluated by the four stakeholders $\{st_a, st_b, st_c, st_d\}$ – evaluations are depicted in Table 6.3.

In order to determine a group recommendation we can apply group decision heuristics [Mas11]. For example, the *majority voting* strategy (see Table 6.3) recommends a value that represents the majority in the set of individual votes. Another example of such a group decision scenario is the assignment of requirements to software releases. In this context as well stakeholders can have different preferences regarding the assignment of a requirement to a specific release. When applying majority voting (MAJ), the release with the highest number of votes would be assigned to the corresponding requirement. As a result of a couple of empirical studies [FN12], majority voting (MAJ) has been selected and integrated as the primary decision heuristic of the IntelliReq environment. In addition to the analysis of individual decision heuristics, [FN12] also introduce a meta-heuristic that combines individual heuristics into an ensemble. Ensemble-based heuristics showed to outperform individual heuristics [Nin12] and therefore will be integrated and evaluated in new versions

[2] http://www.openthesaurus.de

[3] The inclusion of English thesauri such as WordNet (wordnet.princeton.edu) is within the scope of future work.

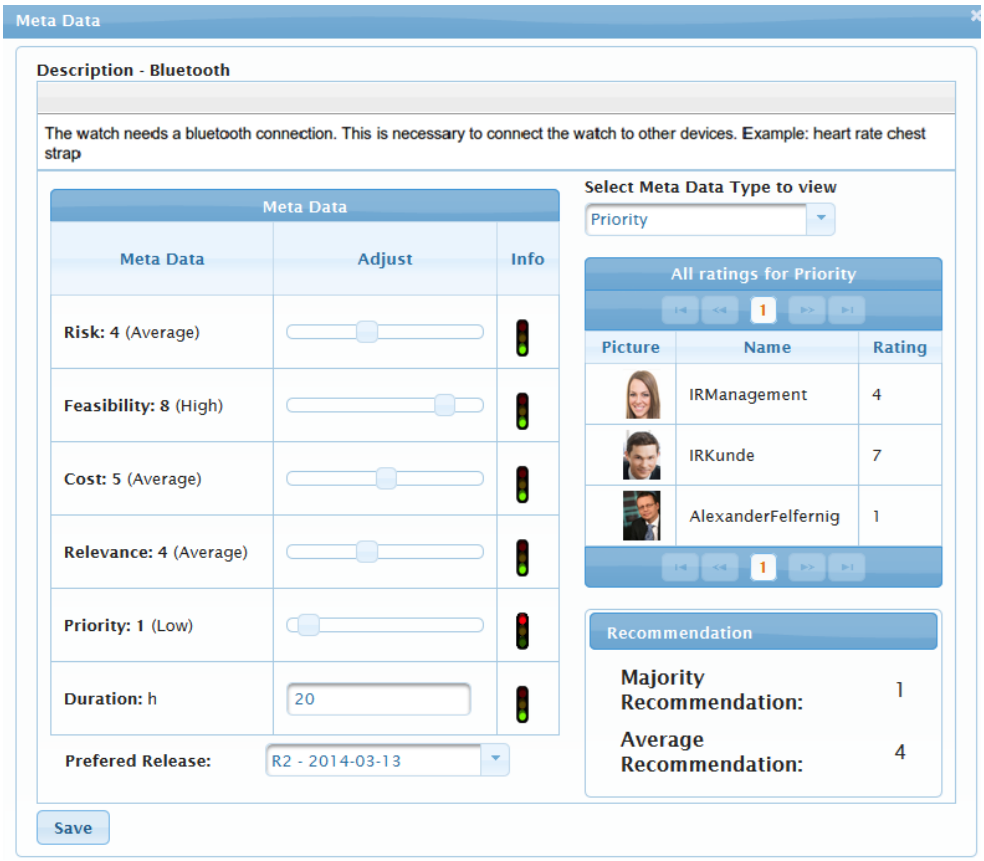

Figure 6.1: IntelliReq: details regarding a single requirement. The three stakeholders provided inconsistent ratings for the property *priority* which is indicated by the traffic light feedback mechanism.

of IntelliReq.

Knowledge-based Recommendation. Knowledge-based recommendation [Bur00][FB08] exploits deep knowledge about items, user requirements and preferences, and their relationships. Recommendation knowledge is represented in terms of constraints (rules) which indicate the relationship between user requirements/preferences and the given item set. This type of knowledge representation supports the generation of explanations as to why items are recommended or no solution could be found [FSR13].

In the Requirements Engineering context, knowledge-based recommenders can be used, for example, for the recommendation of open issues. In Figure 6.1 the three stakeholders have diverging estimates regarding the *priority* of the requirement – this situation can be automatically detected by constraints that indicate open issues to be solved (using the traffic light semantics).

Furthermore, knowledge-based recommenders can be applied in the context of release planning. In IntelliReq release plans are manually defined by stakeholders. Inconsistencies between stakeholder preferences can be repaired on the basis of heuristic search based diagnosis. In addition, we have developed concepts that allow a model-based diagnosis (MBD) of inconsistencies [FSR13][Rei87]. MBD identifies a minimal set of changes in the requirements model such that consistency can be restored. In the case of incomplete release plans (some of the requirements do not have an assigned release), IntelliReq can propose completions that are based on recommendations of group recommendation algorithms [Mas11].

6.3 IntelliReq User Interface

Figure 6.1 and Figure 6.2 provide an impression of the way in which users can define and manage their requirements in IntelliReq. Each requirement has a textual description and is associated with a set of properties (metadata) that describe specific characteristics of a requirement, for example, associated risk, feasibility, and costs. Each stakeholder is encouraged to evaluate requirements with regard to the given set of properties (metadata). For each requirement, IntelliReq provides group recommendations that support a group of stakeholders in deciding about the evaluation of the requirement.

IntelliReq automatically identifies potential dependencies between requirements and determines recommendations that are ranked conform to the degree of similarity between the requirements (see ”support value” in Figure 6.2). In the current version, dependency

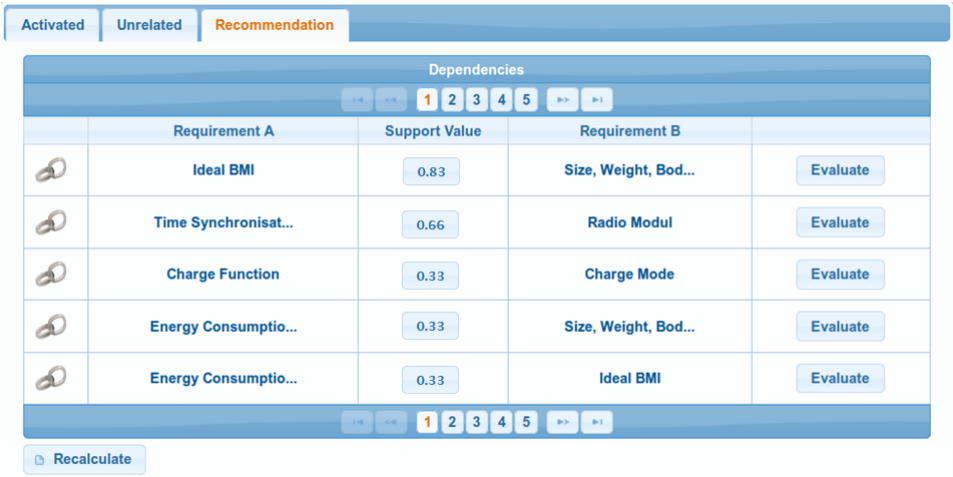

Figure 6.2: IntelliReq: recommendation of dependencies; dependency recommendation is based on OpenThesaurus (www.openthesaurus.de), i.e., IntelliReq currently supports German, the English descriptions used in this chapter have been included for reasons of understandability.

recommendations can be selected and (manually) transformed into corresponding formal dependencies (e.g., *requires* and *incompatible*) that are taken into account as constraints [Tsa93] in release planning.

An important functionality are traffic lights which summarize open issues in an requirements model. For example, if stakeholders evaluate requirement properties differently (e.g., requirement r is considered as infeasible by stakeholder A but completely feasible by stakeholder B), then the corresponding traffic light is red which points out that additional evaluations are needed. In the current version of IntelliReq, a red light is displayed if the corresponding user evaluation exceeds the standard deviation, an orange light is used to point out a low number of stakeholders (less than two) who took a look at the requirement, otherwise a green light is shown.

In IntelliReq, traffic lights are included on different *levels*: (1) contradicting evaluations on the level of requirement properties, (2) neglected requirements in the context of quality assurance (e.g., a requirement has never been evaluated by a stakeholder), (3) unexplained decisions for release plans, and (4) effort-related inconsistencies in the current release plan (e.g., due to too many requirements in a specific release). If the overall implementation effort of requirements assigned to a release is too low or too high, this situation is reflected in terms of red or yellow lights (see Table 6.4).

Table 6.4: Constraints related to the allowed implementation effort of requirements assigned to a release ($RS = \frac{actual\ effort}{allowed\ effort}$).

RS	green	yellow	red
<90%		x	
90-100%	x		
>100%			x

6.4 User Studies and Benefits

In order to analyze improvements that can be achieved by INTELLIREQ, we conducted different system evaluations that will be discussed in the following. First, we analyzed the usability of the INTELLIREQ user interface. Second, we evaluated different INTELLIREQ recommendation approaches.

Usability. This study has been conducted at the Graz University of Technology. N=20 subjects (85% male and 15% female) interacted with the INTELLIREQ environment and developed a requirements model (set of requirements) for an application domain they could choose on their own. In a second step the subjects had to switch to a predefined example set of requirements (digital watch) and to complete a predefined set of tasks such as defining dependencies between the given requirements (with the support of the INTELLIREQ dependency detection) and evaluating meta-properties (e.g., risk level, feasibility, and costs) of requirements. After having completed these tasks, the participants had to fill out a questionnaire based on the system usability scale (SUS) and to answer further questions regarding the applicability of the INTELLIREQ environment. The subjects of the study agreed on the applicability of the system. INTELLIREQ is easy to use and the majority of the subjects stated that they are willing to use the system on a regular basis (see Figure 6.3).

Recommendation Support. Further feedback provided by the subjects of the usability study was the following. Content-based dependency recommendations were appropriate and helped to increase the quality of requirement models (average evaluation 4.35).[4] Content-based recommendation algorithms also alleviated the search for and the reuse of requirements (4.26). Recommendations regarding quality assurance tasks (in terms of a traffic light signal) are helpful and should be constantly shown to the user (4.22).

The outcome of previous evaluations (see [Fel+12]) was that group recommendation increases the perceived system usability and quality of decision support. In this context it is

[4] 1 = I do not agree, 2 = I partially agree, 3 = I rather agree, 4 = I agree, 5 = I totally agree.

important to not disclose individual preferences of group members in early phases of a decision process. The reason for this is that the knowledge about the preferences of other group members leads to an insufficient exchange of decision-relevant information. In future IntelliReq versions we will make this property configurable, i.e., if the administrator prefers to disclose the preferences (evaluations) of different users from the very beginning, she/he will be able to do so. Finally, recommendations to groups intensify discussions between group members which itself has a positive impact on the quality of the decision outcome [Fel+12]. The reason for this is that discussions between group members increase information sharing which itself increases requirements-related knowledge of group members and thus improves the quality of the information needed for taking a decision. In addition to these evaluation

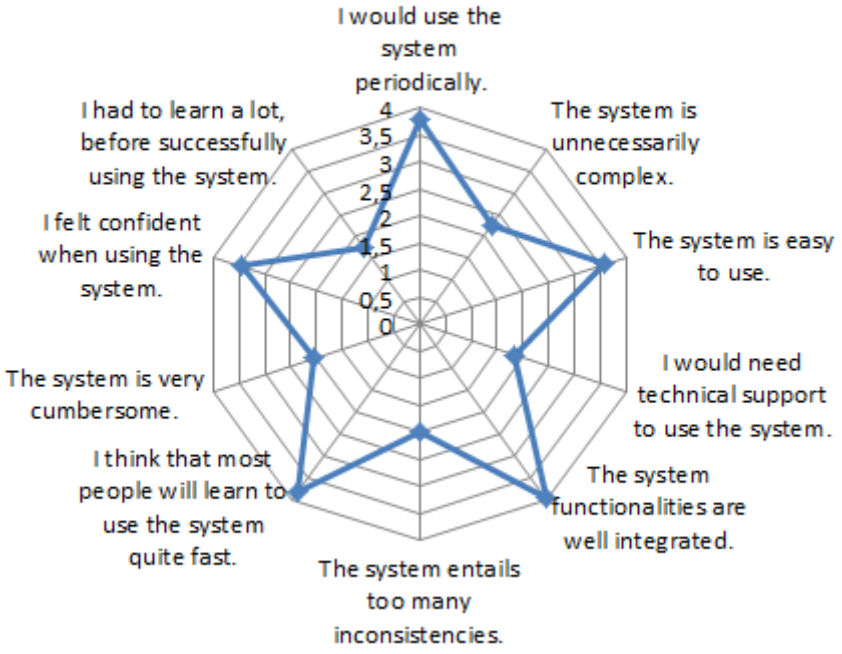

Figure 6.3: SUS usability evaluation: average ratings, N=20 (1 = I do not agree, 2 = I partially agree, 3 = I rather agree, 4 = I agree, 5 = I totally agree).

results we were interested in the impact of recommendations (traffic light support) on quality assurance practices. For example, if less than two other stakeholders (not the originator of the requirement) took a look at a specific requirement, a yellow traffic light is shown (a red traffic light is shown if no other stakeholder took a look at the requirement). This is a kind of knowledge-based recommendation where a constraint specifies the status of the traffic-light. From the psychological point of view people prefer situations where things are complete and they do not need to think about these things any further (completion directly leads to a sense

of having achieved closure). Contrary to this, incomplete things leave us unsatisfied and we seek to resolve the existing incompletion.

In order to analyze the impact of traffic lights based user guidance we conducted an empirical study with N=32 computer science students (22% female and 78% male).[5] In the role of a release manager their task was to analyze an existing requirements model, resolve inconsistencies in the model, and to generate a corresponding release plan. The outcome of this study was the following. When supported by traffic light based recommendations regarding quality assurance tasks (50% of the subjects received such recommendations), users needed significantly less interaction steps (e.g., in terms of the number of changes of the requirements view) ($p < 0.01$) and less time ($p < 0.02$) to successfully complete the given task. Traffic light based indication of tasks also persuaded subjects to document their decisions regarding a release plan ($p < 0.01$).

The semi-automated detection of dependencies between requirements in IntelliReq is based on content-based filtering where the requirements most similar to the requirement currently under investigation are presented to the user. The underlying assumption is that similarity between requirements is an indicator for dependencies. In order to evaluate the quality of the current dependency detection approach (see Section 6.2), we measured precision as an indicator of prediction quality (see Formula 6.2). The average precision (stakeholders *accepted* this recommendation as a dependency) measured in the current projects is 0.692 for the 10 top-ranked requirements (those with the highest support value). Note that even for very small projects with about 100 requirements, the theoretical number of pairs to be analyzed with regard to dependencies is 4.950.

$$precision = \frac{|accepted(req_i)|}{|recommended(req_i)|} \quad (req_i \in REQ) \tag{6.2}$$

Summarization of Benefits. The *major benefits* of the IntelliReq Requirements Engineering environment are the following. *Time efforts* related to the development and quality assurance of requirements can be reduced due to a more *systematic approach of quality assurance* (traffic light based indication of open issues) and due to a *group recommendation support* that helps to mediate between different stakeholders (e.g., in the case of contradicting evaluations of requirements). A further reason for time savings is the *recommendation of potential dependencies* between requirements which otherwise would have to be figured out manually. A more systematic analysis of dependencies can also cause a reduction of inconsistent definitions in the requirements model. In the same line, the support of *requirements reuse* can help to avoid the definition of redundant requirements. From the psychological standpoint,

[5] The subjects of this study did not participate in the usability study.

the traffic light based indication of open issues exploits the phenomenon *need for completion* and thus increases individual *user engagement*.

6.5 Related and Future Work

Recommender Systems in Software Engineering. The application of recommendation technologies in Software Engineering is manifold and ranges from method call recommendations in software development [Tsu+05] to the recommendation of effort estimation methods in project management [Pei+10]. An overview of the application of recommendation technologies in Software Engineering can be found in [RWZ10]. A detailed overview of the application of recommendation technologies in Requirements Engineering can be found in Chapter 2. In the remainder of this section we focus on topics related to recommender systems in the context of IntelliReq.

Stakeholder Recommendation. Crucial for the success of a project is the inclusion of the right representatives of a group. The StakeNet approach [LQF10] supports stakeholder identification on the basis of the concepts of social network analysis. StakeNet social networks are build from individual stakeholder recommendations (e.g., *A* recommends *B* to be part of the project). An example of a network analysis operation in this context is *betweenness centrality* which counts for a specific stakeholder *st* the number of shortest paths between other stakeholders in which *st* is included. A corresponding high value indicates a person's capability of acting as a broker between groups. The inclusion of stakeholder recommendation mechanisms into the IntelliReq environment is an issue for future work.

Recommendation of Requirements. An approach to requirements reuse is presented in Dumitru et al. [Dum+11]. Reuse support is implemented on the basis of content-based filtering where keywords extracted from the description of the new project are matched with keywords extracted from requirements descriptions of already completed projects. In contrast to IntelliReq no Thesaurus information is used when determining content-based recommendations. Furthermore, in contrast to existing approaches to include recommendation techniques in Requirements Engineering processes, no group recommendations (e.g., in the context of requirements definition and release planning) are supported.

Consistency Management. Especially for informal requirements an automated consistency management is unrealistic [IR04]. However, semi-automated approaches as implemented in IntelliReq can help to reduce related efforts. IntelliReq provides a couple of techniques that help to improve consistency management processes. Inconsistencies can, for example, be resolved on the basis of the concepts of model-based diagnosis [Rei87] combined with

corresponding repair algorithms [FB08]. A discussion of the automated diagnosis of inconsistent requirement models can be found in [Fel+10a].

Requirements Prioritization. Restrictions regarding available resources often require prioritization decisions regarding the set of requirements that should be implemented [Dav03]. In disaster scenarios victims are categorized into three types: those who will die, those who will survive, and those whose survival depends on the medication (also known as *triage*). Requirements prioritization is similar: requirements that *must not be included* in the next release, those that are *optional* for the next release, and those that *must be included*. In IntelliReq, requirements prioritization is implemented as a group decision process where for each requirement the group as a whole has to develop a consensus regarding the prioritization (not included, optional, must be included).

Future Work. (1) Requirements Engineering environments are often based on the assumption of stable stakeholder preferences (e.g., regarding the prioritization of requirements). In fact, decision processes in most cases follow a process of incremental preference construction and are subject to different types of biasing effects. Being able to take into account related decision psychological theories requires a strongly interdisciplinary research approach. (2) In order to further improve the quality of dependency detection mechanisms in IntelliReq, approaches from natural language processing [FS05] and text mining [WF05] have to be combined with content-based approaches currently included in IntelliReq. (3) The IntelliReq user interface will be improved in terms of integrating functionalities to automatically annotate and group requirements. (4) In future versions of IntelliReq we intend to provide interfaces to existing Requirements Engineering tools such as IBM Doors.

6.6 Conclusion

Existing Requirements Engineering tools primarily support the definition and cataloging of requirements but fail to provide additional information such as similarity of requirements, dependencies between requirements, and quality status of requirements. In this chapter we presented the IntelliReq environment which focuses on the integration of recommendation technologies with the goal to make Requirements Engineering environments more proactive. Among the major advantages that can be expected from the application of recommendation technologies in Requirements Engineering are an increased reuse of requirements, active guidance of stakeholders, increased consistency in requirements models, and reduced time efforts needed for the construction of requirement models.

7 Future Work: Human Computation in Requirements Engineering

We already motivated the need of a semi-automatical evaluation of requirements represented in natural language (see Chapter 4) and proposed an approach based on the exploitation of lexical resources. However, for a satisfying interpretation of requirements written in natural language, one needs commonsense knowledge which is often omitted in written requirements descriptions as it is assumed that this kind of information is already known to a potential reader. As a simple example, you cannot infer from the lexical resource that if your *cat is sick* you should visit a veterinarian. Of course, there are approaches to collect these relations in a database to infer these kind of conclusions [Ahn05]. However, having a database containing that vast amount of information to achieve a human like reasoning is implausible. A founder of AI, Marvin Minsky, once estimated that knowledge about 30 to 60 million things and the power to make analogies based on them is necessary for commonsense reasoning [LS04]. Thus, humans are good at finding out the context of words as they own the necessary knowledge on these things and can retrieve this information quickly [LS04]. As a logical consequence enhancing natural language processing with human interaction can be of high value for the quality of the result.

In this chapter we want to discuss the possible exploitation of the idea of Human Computation in the scope of Requirements Engineering. Human Computation in the modern usage appears to be inspired by von Ahn's 2005 dissertation titled "Human Computation" and defines the usage of human processing power for problems which cannot be solved by computers yet [QB11][Ahn05]. Our interest is focused on two main problem areas: *ambiguity* in requirements descriptions and the need for domain-specific ontologies to automatically discover *inconsistencies and incompleteness in requirements specifications*.

7.1 Ambiguity

Unrecognized ambiguity occurs if a reader interprets a statement in a requirement with the first meaning coming to her/his mind without being aware of other possible meanings. This can lead to a completely wrong understanding of the given requirement and is a major source

of failure in software projects [BK04]. To reduce this problem we propose a gaming approach to increase the discussion of meaning of terms used in projects and / or in the domain context. To do so we want to adapt the *ESP* game [Ahn05] originally used to annotate images on the web. During the game two players provide words to describe a picture. If they come up with the same word, they proceed with the next picture and receive some points for successfully completing the given task. The results found with this approach revealed a high precision which outperformed image recognition algorithm available around 2005 [Ahn05].

Although the original approach was designed to find universally valid annotations for images, we are convinced that this approach fits even better into the context of Requirements Engineering. For example, a company wants to create a glossary of terms commonly used in their software projects. In this context, instead of universally valid descriptions for the used terms, the company needs a glossary which reflects the interpretation of terms as it is used in their daily business. This interpretation may even be conflicting with a universally valid view on specific terms. Thus, we assume that knowledge bases generated through games with a purpose [Ahn05] played in a company scope outperform public available general purpose resources like ontologies for tasks related to the company.

Consequently, to create the aforementioned company specific glossary, stakeholder participating in the Requirements Engineering process need to find consensus about the interpretation of terms. In the *EasyWinWin* process this is done during the brainstorming session where stakeholders pair up and try to come up with definitions for key terms. Beside the definition of the correct interpretation of terms this approach also supports the identification of tacit knowledge which would not be communicated otherwise [GB01]. We adopt the idea of pairing up two stakeholders and combine it with the concept of the *ESP* game [Ahn05].

Define input for a game scenario

Opposite to the original *ESP* game, assigning tasks to users pure randomly is not a good approach as we have to deal with different conditions:

- *Less Participants:* The original *ESP* game was played by 13,630 players during a four-month period [Ahn05]. In our context the amount of potential participants in a game is far lower as only persons related to a specific domain are valid players (for example, stakeholders in a project context or employees of the company).
- *Unequal important terms:* Not all terms in a requirement are equally important for a succeeding NLP task. For example, terms like *DNS* or *database* may be more important than *cold* or *cosy* for a software company. Also, to achieve good results each task (find a description for a term) should be repeated several times. Therefore, it is necessary to find intelligent methods to assign the most relevant tasks to players first. Additionally,

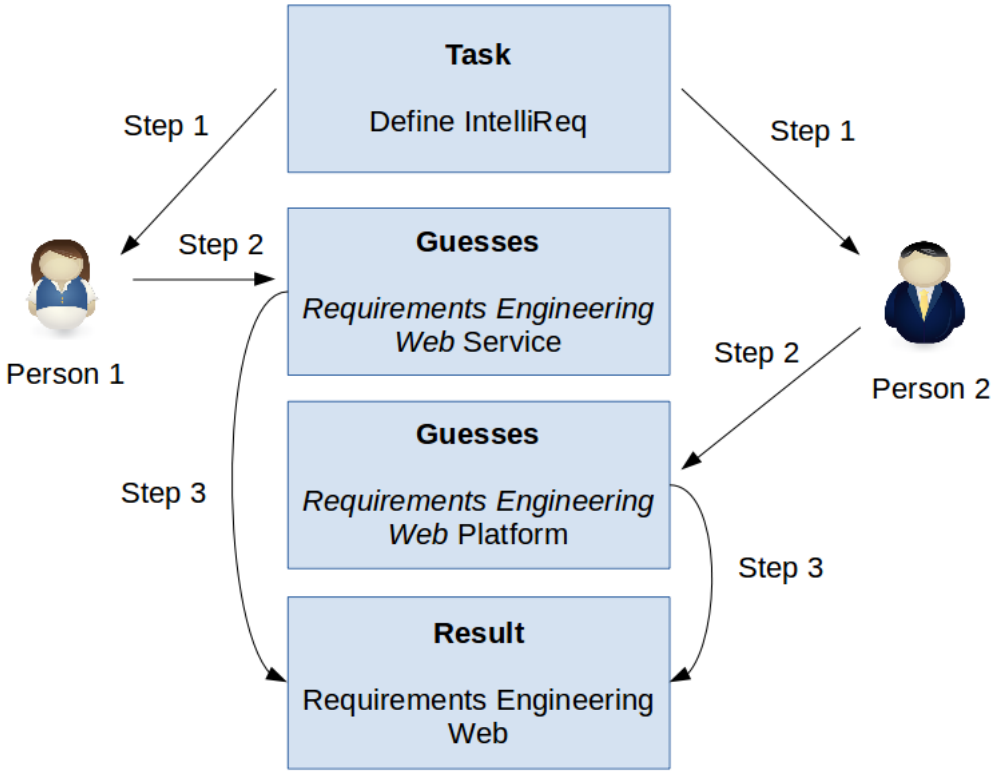

Figure 7.1: AMBIGUITY GUESSES GAME: Shows the different steps to define annotations for the term *IntelliReq*: Step 1 distributes the task of defining the term *IntelliReq*. Step 2 shows the responses of the two participants. The term *Requirements Engineering* is a match between the two responses and is shown in italic for better understanding. Step 3: The system takes the matching term and define it as result of this game round.

assigning meaningless tasks (unimportant terms) may be annoying for the players and would reduce the amount of liberally played games.

- *Probability of ambiguity:* As motivated in Chapter 4 the cardinality of concepts is different between terms and a high cardinality increases the risk of using a term in the wrong context. Consequently, terms with high cardinality should be played more often than other terms.

As a consequence from these issues we cannot randomly assign terms as tasks to the players. Instead, we need to focus on the most important terms first. We therefore propose a filtering to identify the best candidate terms for our game. In a first step, we want to identify company specific terms which could be used for products, processes, or similar. For a straight forward method to identify candidate terms one can take common used words (e.g., words with an occurrence above a certain threshold) from documents created in a company. Next, these

words are compared to a lexical resource like *OpenThesaurus*[1]. If there is no matching entry for a word in the lexical resource we define this word as candidate for a company term. These candidates can be used as input for a succeeding game with the purpose of defining company terms.

Of course, one can assign a knowledge engineer to create an ontology with these terms but there are several advantages coming with our gaming approach. First, if different stakeholders are participating in the process of finding relations between company terms, one can infer the distribution of necessary knowledge about these terms between the stakeholders. For example, having only few stakeholders agreeing on a term will indicate the need for discussion about this term. Second, having a search over all requirements terms and comparing them with lexical resources, the risk of overseeing a company term for a potential ontology is reduced. To illustrate our gaming approach we use the company term *IntelliReq* which is a collaborative platform to facilitate the Requirements Engineering process. Figure 7.1 shows a simple game flow to define the term *IntelliReq*. We start with two players named *Person 1* and *Person 2* with the task to agree on a description for *IntelliReq*. Without seeing the input of the other person, each player provides terms which describes *IntelliReq* best in their opinion (we denote them as *Guesses*). Comparing the provided information the system can identify the term *Requirements Engineering* as a match both player can agree upon. Having at least one matching term, the system considers the game as successfully completed and rewards the player with a certain amount of game points.

As mentioned above, in the scope of Requirements Engineering in a company, there are far less participants available compared to the scenario of the *ESP* game and the response time between *Step 2* and *Step 3* (see Figure 7.1) should be reasonably short. We therefore propose a dynamic approach (see Figure 7.2) which replaces a human player by data collected from previous game sessions after a certain period of time has passed without finding another player.

Based on the experience collected during our study for the *Keyword Recommender* (cf. Chapter 4) there is another topic which must be considered. We argued that a recommendation of keywords can be used to reduce different versions of the same term to annotate a requirement. Similar to that we can identify the terms *Web Service* and *Web Platform* as guesses provided by the player in Figure 7.1. Of course, the system can only use the term *Web* but this is obviously an insufficient tag to describe the *IntelliReq* environment. Also, one can argue that these terms are not equal according to the definition in a dictionary, however in the context of Requirements Engineering it is more important how terms are used in the project context. Consequently, we need to identify how stakeholders in a project context use these terms and if they are a valid option to describe *IntelliReq*.

[1] We used this lexical resource in our implementation of IntelliReq.

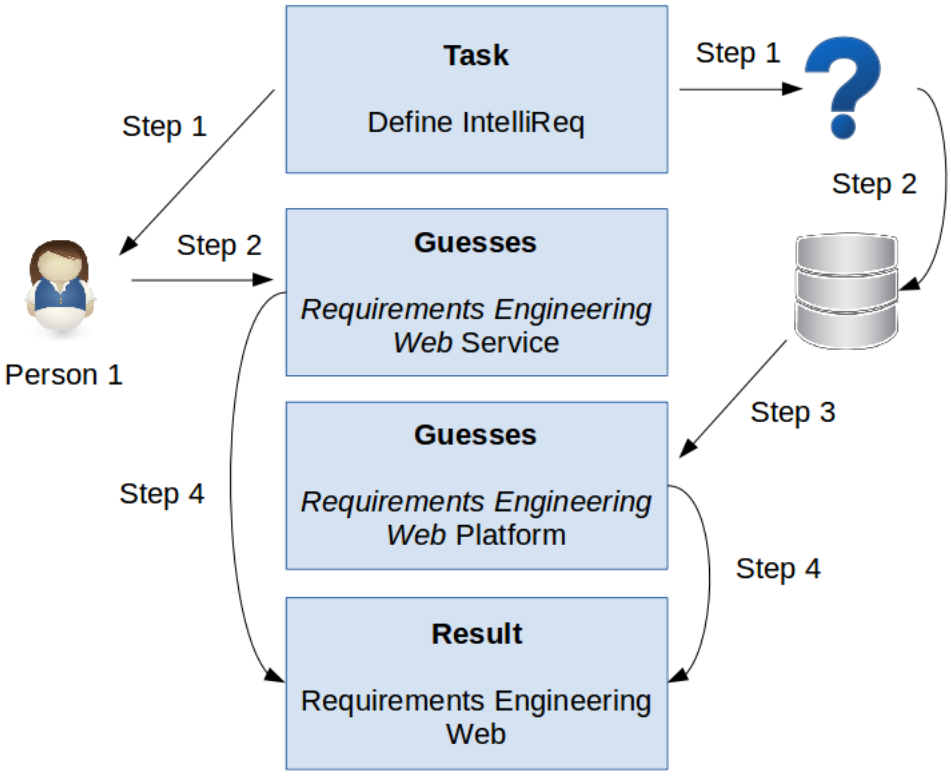

Figure 7.2: AMBIGUITY GUESSES GAME WITH ALTERNATIVE GAME FLOW: Shows the different steps to define annotations for the term *IntelliReq* in case a second player cannot be found in a reasonable time: Step 1 distributes the task of defining the term *IntelliReq*. Step 2 is equal for *Person 1* as shown in the previous approach. Also, the system could not find a second player and associate the player slot with the database. In Step 3 the system takes a random response from the past. Step 4 is equal to Step 3 described in Figure 7.1.

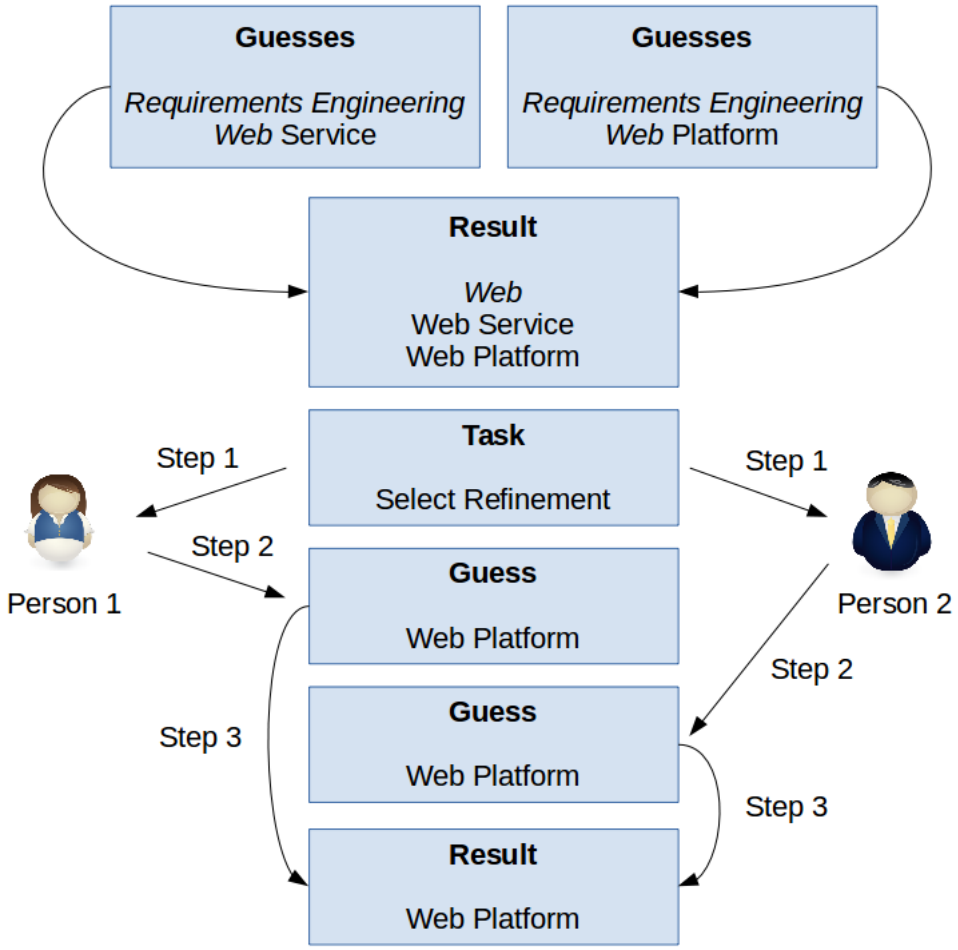

Figure 7.3: AMBIGUITY GUESSES GAME BONUS ROUND: Shows the refinement process for the annotation of the term *IntelliReq* starting with the results of the previous game (cf. Figure 7.1 and Figure 7.2). There are two possible refinements for the term *Web*: Step 1 distributes possible refinements for the term *Web*. In Step 2 both participants guess that the term *Web Platform* is a good refinement. Step 3: The system takes the matching refinement and define it as result of this game round.

To resolve this problem, we propose the use of a bonus round. In this bonus round participants have to agree on a refinement of the given term based on guesses from the past (see Figure 7.3). Note that this bonus round is not necessarily done by players who came up with the initial results from the previous step. Furthermore, it should be the topic of a future study to evaluate if reassigning participants, who came up with the initial results, to bonus rounds increase the quality of tags compared to randomly inviting players to bonus rounds. Beside the identification of company terms we also want to gather additional information during the game, for example, if there are a lot of concepts (cf. Chapter 4) associated to a term. This information can be of high importance for a succeeding natural language processing. For

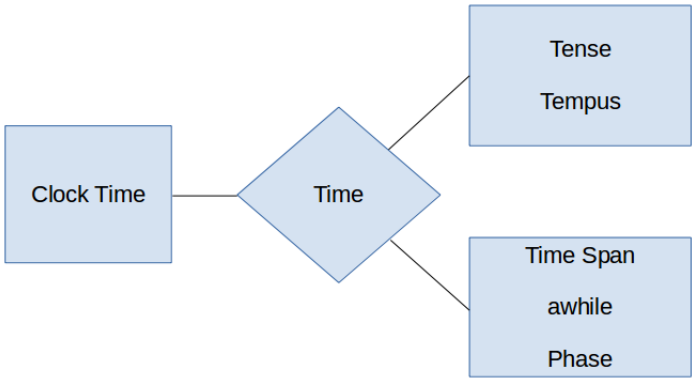

Figure 7.4: CONCEPTS OF THE TERM *Time*: Shows the different possible concepts in which the term *Time* can be used. As can be seen the cardinality of *Time* is three.

example, terms with a high cardinality of concepts can be easily misused during the process of seeking related documents as they are highly dependent on the context they are used in. In our approach in Chapter 4 we are using the cardinality of concepts to reduce the weight of terms for our similarity measurement.

Independent of the process of generating the list of terms which should be evaluated, we propose a game to dissolve the ambiguity of used terms. Opposite to the approach of refining company specific terms, we can exploit information stored in lexical resources. For example, synonyms can be used to propose alternatives for each concept of a term to participants. Confronted with the list of possible options, the participants can try to agree on one synonym of a concept which fits best in the domain of interest. As an example, we use the design of a simple watch. In this context, the term *Time* is under investigation. As can be seen in Figure 7.4 there are three possible concepts available for this term[2].

Using our example set and assuming that both participants can agree on *Clock Time* as best fitting concept for their working environment, the game flow is shown in Figure 7.5. In the context of designing a watch the most useful interpretation of the term *Time* is in thinking of *Clock Time*. Obviously, one can argue that there is also the possibility of thinking about a *Time Span* but there should be no association to the concept of *Tense* or *Tempus*. Table 7.1 shows a possible scenario with four game rounds. Each round ended with one concept found (three times *Clock Time* and one time *Phase*).

We use Formula 7.1 to calculate weights for the concepts with $|concept(i)|$ is the cardinality of game rounds ending with the concept i found. Based on this formula we come up with a weight of 0.75 for the term *Clock Time* and with 0.25 for the term *Phase*. Finally,

[2]These three concepts are taken from the lexical resource *OpenThesaurus* which is also used in our INTELLIREQ environment.

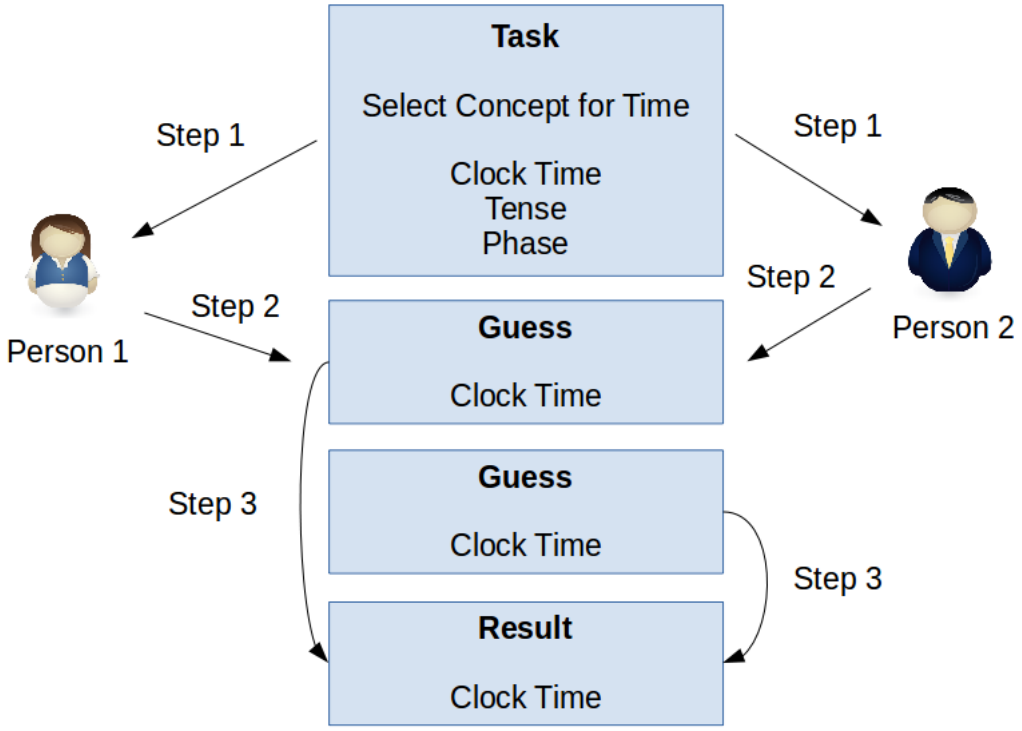

Figure 7.5: CONCEPTS OF THE TERM *Time*: Shows the different possible concepts in which the term *Time* can be used. As can be seen the cardinality of *Time* is three.

as no one voted for the term *Tense* this concept will have a weight of zero and will be ignored in the context of the term *Time*. Thus, if we apply these weights to a succeeding similarity measurement task between requirements we will receive a strong binding between documents about *Clock Time* and *Time* which is reasonable in our scenario. On the other side, requirements about *Time* and *Tense* will not be linked together (at least based on these terms).

It is also possible to use these weights to improve similarity algorithms for text comparison such as the TF-IDF (Term Frequency - Inverse Document Frequency) algorithm which is similar to the approach used in our current INTELLIREQ environment with an important difference. Having no domain knowledge, we used a probability measure based on the cardinality of concepts (cf. Chapter 4). This is based on the assumption that terms with a high cardinality have a high probability to be used in the wrong context.

$$weight_i = \frac{|concept(i)|}{|playedRounds|} \tag{7.1}$$

To improve the quality of our proposed game to find concepts for terms (see Figure 7.5) the

Table 7.1: Example results of the concept game.

Game	Clock Time	Phase	Tense
1	x		
2		x	
3	x		
4	x		

Phetch [Ahn05] game can be used with some modifications. Similar to the original game, a player called *Describer* has to explain a specific term to a second player called *Seeker*. If the second player can correctly guess the term looked for, both players are awarded with a certain amount of points. Also, if it is possible to find the term looked for with the provided description, this can be assumed as proof for the quality of the description. Additionally, by modifying the available options to describe a term (e.g., deny the use of certain words), the quality of the different description terms can be evaluated. As a simple example, Table 7.2 shows four played games. In the first run, the *Describer* proposes the terms *Collaboration Platform* and also needs to show the second term *Requirements Engineering* to the Seeker before he can correctly guess the term *IntelliReq*. In the next run the *Describer* uses the terms *Web Platform* and *Requirements Engineering* and again the *Seeker* can find the term looked for. Obviously, the term *Requirements Engineering* is useful to describe the term *IntelliReq* as it was used in both previous rounds. Therefore, the system does not allow the usage of this term in the next game so that the *Describer* has to come up with an alternative description. In the third game having only the terms *Web Platform* and *Collaboration Platform* available, both players do not succeed in finding the correct result. The reason could be that there exist several web based collaboration platforms in the specific domain and therefore these two terms are insufficient to correctly identify the term *IntelliReq*. Consequently, a description using only *Collaboration Platform* and *Web Service* is insufficient. In the last game run, the players are only allowed to use the remaining term *Requirements Engineering* to describe *IntelliReq*. Again, there could be no result found which could be based on the fact that there are different entities in a domain related to *Requirements Engineering* like desktop applications and processes. Therefore, none of these terms is redundant for the description of the term *IntelliReq* but the term *Requirements Engineering* should receive a higher weight as it is involved in all successful game runs.

7.2 Relations between Objects in an Area of Interest

Although natural language processing has advanced, it is hard to handle informal descriptions in Requirements Engineering documents [KS06]. We also motivated in Chapter 4 that the

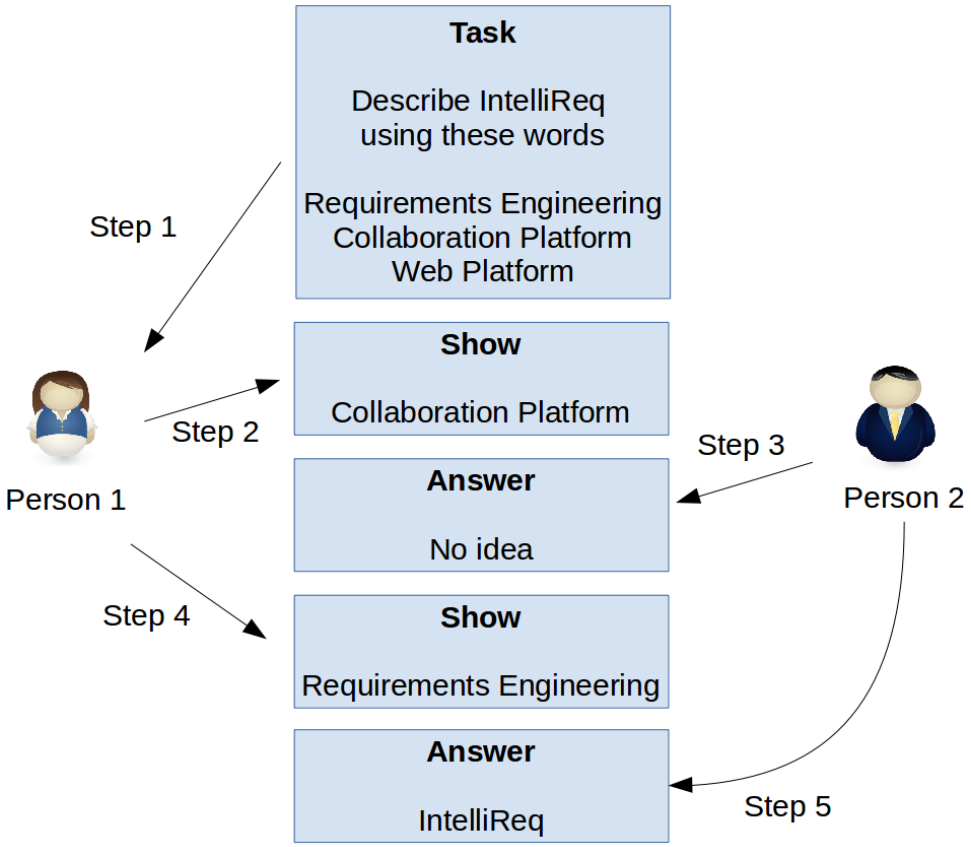

Figure 7.6: CONCEPTS OF THE TERM *Time*: Shows the different possible concepts in which the term *Time* can be used. As can be seen the cardinality of *Time* is three.

Table 7.2: Example game results for the term *IntelliReq*. An empty field in the column *Second description* indicates that the correct term was found without the need for a second description shown to the other player.

Game	First description	Second description	Result
1	Collaboration Platform	Requirements Engineering	IntelliReq
2	Web Platform	Requirements Engineering	IntelliReq
3	Collaboration Platform	Web Platform	No result
4	Requirements Engineering	-	No result

Table 7.3: Example game results for the term *IntelliReq*. An empty field in the column *Second description* indicates that the correct term was found without the need for a second description shown to the other player.

Type	Task	Answer
is a	*IntelliReq* is a ...	Collaboration Platform
requires	*IntelliReq* requires ...	Web Server
is used for	*IntelliReq* is used for ...	Requirements Engineering

use of a semi-formal notation is often not possible as you cannot assume all stakeholders to be capable of using formal notations. Thus, Requirements Engineering processes are often bound to natural language but can be supported by lightweight semantic processing based on a domain ontology to detect inconsistency or incompleteness in requirements descriptions [KS06]. An ontology is a specific set of objects, concepts, and other entities which exist in an area of interest. It also contains relations between all these items. Thus, it represents a simplified view of the world that fits a specific purpose [Gru93]. This definition leads to the necessity of defining an own ontology for the use in NLP within a specific project and / or company context.

With a domain-specific ontology we can find pointers to concepts. Between atomic terms there must be connections such as *requires*, *is part of*, aso. With that we can identify properties of requirements which help to detect incompleteness and inconsistency. For example, if the ontology defines the term *reservation requires cancel*, a requirement can be defined as incomplete if it only mentions reservations without an option for cancellation. There is a need to find these relations [KS06].

Similar to the task of finding annotations for terms used in a specific domain, we can use the proposed game flow in Figure 7.1 to construct relations between terms. We only need to change the task from *Define IntelliReq* to one of the question types shown in Table 7.3. Again, the goal of the game is that two participants come up with the same result for the given task and are rewarded with game points if they succeed.

8 Conclusion

8.1 Limitations of the INTELLIREQ Environment

The following section shows known limitations of the INTELLIREQ environment and suggestions for future enhancements.

Keyword recommendation is currently only based on keywords used for requirements in the past and does not provide any evaluation of the quality of recommended keywords. Thus, if other users annotated requirements with mismatching and / or misspelled keywords, these words will be recommended if they match with words found in a new created requirement description. In our study, we provided an initial set of keywords suitable for the given software development task. As shown in Chapter 7 the generation of such an initial set of keywords can be done collaboratively by the use of games with a purpose which should be supported in a future version of INTELLIREQ. Also, there is a need for a better support during the keyword insertion process, e.g., a spell checker should be provided by the system to prevent typos in the set of keywords.

Dependency detection in INTELLIREQ is done by comparing the occurrence of synsets (synonymy sets) in two different requirements. As a consequence, the recommendation quality of potential dependency candidates relies on the relevance of synsets used for this comparison in the domain of interest. Currently, only the cardinality of possible meanings of words is taken into account. To improve this approach a knowledge base with domain relevant keywords (and related synsets) can be used. With this knowledge base, if two requirements descriptions share a synset which is defined as domain relevant, the similarity score should be increased.

Evaluation of the quality of requirement descriptions is only supported with a very simple rule (how many stakeholders viewed a requirement). Although, this is a good starting point, an improved environment should support a more intelligent approach. For example, having the history of viewed requirements by stakeholders, INTELLIREQ could use collaborative filtering to recommend specific requirements to stakeholders. For example, two stakeholders with the same history of viewed requirements may posses the same expertise in a specific area.

As a consequence, if one stakeholder evaluated a requirement the system should recommend this requirement to the second stakeholder with a similar expertise.

8.2 Conclusions from Research Questions and Contributions

To support stakeholders in the process of defining complete and consistent requirements documents, computer-aided support is required. In the following we reflect on our research questions and contributions.

Research Question Q1:

How do recommendations and preference visibility influence the perceived usability and quality of decision support in Requirements Engineering environments?

In this work we conducted a study with 293 participants to investigate the impact of recommendation technologies applied to the field of Requirements Engineering. For this purpose we developed a prototype environment where stakeholders could define, adapt, and store preferences. Also, it was possible to discuss already defined preferences and to receive a recommendation for group decisions related to the task of finding requirements for a software project. For the evaluation we created a simple interface in which seven basic decisions had to be defined (see Section 3.2) and divided the study participants into four different groups: with preference view and recommendations, with preference view and without recommendations, without preference view and with recommendations, and without preference view and without recommendations. The results were the following: although the usability does not seem to increase through the use of recommendation technologies, the perceived decision support is better. Additionally, group recommendation technologies seem to be a stimulating element for information exchange and visible preferences reduce the amount of adapted preferences as stakeholders want to be seen as consistent with their original preferences. We detected a decision bias when stakeholders were confronted with the preferences of other stakeholders as they base their initial preferences on the information provided by other stakeholders. Unfortunately, this leads to less information exchange which is very important for the quality of the decision making process. Finally, we identified a decreased satisfaction if someones preferences are not taken into account which leads to a win-lose situation.

Research Question Q2.1:

How to increase stakeholder interaction with requirements and release plans to improve the quality of these artifacts?

To guide stakeholders through the Requirements Engineering process we introduced a traffic light based status indication. For example, if only one stakeholder was involved in the definition of a requirement and this requirement is not reviewed by another stakeholder, this can lead to several issues with the descriptions. For example, important information can be left out or the description itself is ambiguous. We therefore defined a minimum threshold of involved stakeholders to assure the quality of the description of requirements. To motivate stakeholders we used a yellow traffic light near a requirements description which has not reached the minimum number of reviews by other stakeholders. In a similar way we used traffic lights to indicate flaws in release plans for a set of requirements. We therefore defined an amount of work hours available for each release and all requirements possessed hour estimations for their implementation. Based on this information we defined three different conditions. First, one assigned too less requirements to a release and therefore did not use the available work hours. This condition was indicated by a *yellow traffic light*. Second, a *green traffic light* was used if stakeholders assigned requirements with a total working time between 90% and 100% of the available hours of a release. Finally, if stakeholders assigned too many hours to a release this was indicated with a *red traffic light*. Additionally, stakeholders were able to leave an explanation after they finished the release plan. If this comment was missing, we indicated this with a yellow traffic light. In this thesis we provided a study (see Section 6.4) with the following outcome: when supported with traffic lights, stakeholders need less interaction steps to find requirements with the need for further investigation and, as a consequence, can complete the task of release planning faster than stakeholders without traffic light support. Also, we discovered an increase of explanations for release plan decisions in the group with traffic light support.

Research Question Q2.2:

How to increase stakeholder communication about property decisions?

In our latest version of IntelliReq we introduced an additional *Meta Data* view (see Section 6.2.2) to extend the description of requirements. Within this view stakeholders are able to provide their preference information for requirements related to six different dimensions: *risk, feasibility, cost, relevance, priority*, and *preferred release*. This information is stored in a non-anonymous way and is visible for other stakeholders engaged in the Requirements Engineering process. Consequently, stakeholders can discuss with each other

(directly) about their preferences and share their knowledge about requirements.

According to our findings in Chapter 3, increased commitment caused by the visibility of preferences of other stakeholders decreases the consensus in the decision making process. As a consequence, it is necessary to provide analytical processes for the decision making, which is why we added group recommendations to the *Meta Data* view to ease the finding of a consensus.

To motivate the interaction with the *Meta Data* view we used traffic light indicators with different conditions: a red traffic light (strong dissent) and a yellow traffic light (weak dissent) indicated the degree of dissent between the involved stakeholders related to a specific dimension, e.g., the costs of a requirement. In case of a dissent, stakeholders need to start a discussion with other involved stakeholders with the goal to find a consensus. After the discussion changes to the initial preferences are stored by each stakeholder in the *Meta Data* view in the IntelliReq environment. This will trigger a recalculation of the degree of dissent and will set the affected traffic light to green in case the stakeholders could agree on a consensus. Based on the assumption that stakeholders want to finish the Requirements Engineering process without any dissent regarding the importance of requirements, the traffic light annotation will increase stakeholder communication about properties with the goal to resolve existing conflicts.

Research Question Q2.3:

How to identify candidates for dependency relations between requirement descriptions?

We introduced a *Dependency Recommender* which uses information about synonym relations stored in the lexical resource *OpenThesaurus*. We also used the cardinality of *Word Senses* as an easy way to reduce the influence of words with a high probability of being used in the wrong context. A major benefit of our approach is the fast identification of dependency candidates allowing the computation of recommendations in real time. For the evaluation of our approach we used a requirements set (n=30) created for a *sports watch* in an independent brainstorming session. It should be noted that this brainstorming session did not include any considerations about potential dependencies between the created requirements. With our recommendation technique we calculated a ranking of all 435 possible combinations between the requirements. We took the first 20 dependency candidates (with the highest ranking) and presented them as recommendation to participants of a study. During the evaluation we detected a high acceptance of the recommended dependencies.

Research Question Q2.4:

How to increase the reuse of requirements in software projects?

In this thesis we introduced a keyword recommendation technique for requirements (see Section 4.4.1) which was used to support stakeholders in the task of finding a suitable annotation for new requirements. A major goal of the recommendation was the reduction of the amount of different keywords used to annotate requirements, e.g., instead of *hotels* the keyword *hotel* was recommended if another requirement already was annotated by this keyword. For this reason we provided a central storage of all keywords used in the past to annotate requirements in a study group. Next, whenever a new requirement was created we compared all words in the description with the existing set of keywords from the central storage. In case we found a match between the requirements description and the set of keywords we recommended the matching word as keyword for the new requirement. We evaluated our recommendation approach in a study conducted at our university with 39 different software development teams and could proof that this recommendation technology significantly reduced the amount of different keywords used to annotate the set of requirements in the central repository[1]. To evaluate the impact of the reduced amount of different keywords, participants were confronted with the task of creating a set of requirements for a software project and were encouraged to reuse existing requirements from the central repository. As a result of our study we could detect a significant increase of reuse activity in the study group with access to the *Keyword Recommender*.

Research Question Q2.5:

Does preference visibility influence the quality of the Requirements Engineering process?

To evaluate the impact of visible preferences we conducted a study (see Section 5.3) where participants had to develop a software component with an average effort of about 8 person months. We investigated the impact of preference visibility on different dimensions. One dimension is the degree of dissent[2] considering the preferences about the different requirements distributed among the members of the development team. Another dimension was the output quality which has been derived from criteria such as degree of fulfillment of the specified requirements. Finally, we introduced *decision diversity* which is defined as follows:

[1] Although the amount of different keywords used for the annotation of all requirements in the repository was lower, the requirements themselves did not have a lower amount of keywords. In fact, the same keywords were used more often to annotate different requirements than in groups without our *Keyword Recommender*.

[2] Which is the inverse function of the consensus about the preferences of requirements.

if all requirements of a software project have the same priority, the decision diversity is zero. On the other hand, the maximum decision diversity occurs between two requirements if one has the highest possible priority (on the available scale) and the other the lowest possible priority. Our study showed three results: (1) anonymous preferences increase the consensus between stakeholders in software development teams about preference decisions. (2) Anonymous preferences increase the diversity of decisions taken during the prioritization process. (3) Anonymous preferences lead to an increased output quality of the software product being developed.

Research Question Q2.6:

What is the prediction quality of group recommendation heuristics in a Requirements Engineering scenario?

In Section 5.4.2 we introduced three new heuristics: *Median Based*, *Ensemble*, and *Standard Deviation Based* with the goal to improve the prediction quality of group decision heuristics in the Requirements Engineering context. For our evaluation we conducted a study where participants had to collaboratively prioritize requirements for a software component. We used the collected preferences and decisions taken by the teams to calculate the prediction quality of well known group decision heuristics [Mas11] and to compare them with the three new heuristics. Our study revealed that the *Ensemble* and the *Standard Deviation Based* heuristics have the highest prediction quality. However, the *Standard Deviation Based* heuristic was defined for the specific dataset (it uses the deviation of all collected preferences and decisions), which is why it is out of competition.

Research Question Q3

How to use the idea of Human Computation to support Natural Language Processing in the context of Requirements Engineering?

In Chapter 7 we presented ideas for future work to exploit *Human Computation* in the context of Requirements Engineering. In Section 7.1 we described an adaption of the *ESP* game [Ahn05]. We discussed differences between the original game setting presented by von Ahn [Ahn05] and the Requirements Engineering context. Additionally, we motivated another game mode in Section 7.2 to identify relations between objects in an area of interest.

Appendix

Bibliography

[AB06] A. Andreevskaia and S. Bergler. "Mining WordNet for a fuzzy sentiment: Sentiment tag extraction from WordNet glosses." In: *EACL*. Vol. 6. Trento, Italy, 2006, pp. 209–215 (cit. on p. 50).

[Ahn05] L. Von Ahn. "Human computation." PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University, 2005 (cit. on pp. 3, 8, 12, 13, 97, 98, 105, 114).

[AM07] S. Anand and B. Mobasher. "Contextual recommendation." In: *From web to social web: Discovering and deploying user and content profiles* (2007), pp. 142–160 (cit. on p. 31).

[AP05] B. Alenljung and A. Persson. "Decision-making activities in the requirements engineering decision processes: A case study." In: *ISD 2005*. Karlstad, Sweden, 2005, pp. 707–718 (cit. on pp. 4, 33, 38, 44, 71).

[AW03] A. Aurum and C. Wohlin. "The fundamental nature of requirements engineering activities as a decision-making process." In: *Information and Software Technology* 45.14 (2003), pp. 945–954 (cit. on pp. 1, 21, 33, 66).

[BFG11] R. Burke, A. Felfernig, and M. Goeker. "Recommender systems: An overview." In: *AI Magazine* 32.3 (2011), pp. 13–18 (cit. on pp. 15, 30, 83, 84).

[BGB01] B. Boehm, P. Grünbacher, and R. O. Briggs. "Developing groupware for requirements negotiation: Lessons learned." In: *IEEE Software* 18.3 (2001), pp. 46–55 (cit. on pp. 34, 38, 42–44, 66, 70).

[BI96] B. Boehm and H. In. "Identifying quality-requirement conflicts." In: *IEEE software* 13.2 (1996), pp. 25–35 (cit. on p. 62).

[BJP91] J. R. Bettman, E. J. Johnson, and J. W. Payne. "Consumer decision making." In: *Handbook of consumer behavior* 44.2 (1991), pp. 50–84 (cit. on p. 66).

[BJP98] J. R. Bettman, E. J. Johnson, and J. W. Payne. "Constructive consumer choice processes." In: *Journal of Consumer Research* 25.3 (1998), pp. 187–217 (cit. on pp. 4, 22, 33, 38).

[BK04] D. M. Berry and E. Kamsties. "Ambiguity in requirements specification." In: *Perspectives on software requirements*. Springer, 2004, pp. 7–44 (cit. on p. 98).

[Bly02] J. Blythe. "Visual exploration and incremental utility elicitation." In: *AAAI/IAAI*. Edmonton, Alberta, Canada, 2002, pp. 526–532 (cit. on pp. 4, 34, 44, 71).

[Boe88] B. Boehm. "A spiral model of software development and enhancement." In: *Computer* 21.5 (1988), pp. 61–72 (cit. on p. 66).

[BP84] V. R. Basili and B. T. Perricone. "Software errors and complexity: An empirical investigation." In: *Communications of the ACM* 27.1 (1984), pp. 42–52 (cit. on p. 1).

[BT12] C. Bouras and V. Tsogkas. "A clustering technique for news articles using WordNet." In: *Knowledge-Based Systems* 36 (2012), pp. 115–128 (cit. on p. 49).

[Bur00] R. Burke. "Knowledge-based recommender systems." In: *Encyclopedia of Library and Information Systems* 69.32 (2000), pp. 180–200 (cit. on pp. 15, 16, 23, 27, 84, 85, 89).

[Bur02] R. Burke. "Hybrid recommender systems: Survey and experiments." In: *UMUAI* 12.4 (2002), pp. 331–370 (cit. on pp. 2, 15, 84).

[CA07] B. H. C. Cheng and J. M. Atlee. "Research directions in requirements engineering." In: *2007 Future of Software Engineering*. IEEE Computer Society. Minneapolis, USA, 2007, pp. 285–303 (cit. on pp. 2, 6, 47–49).

[Cas+08] C. Castro-Herrera et al. "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)*. Barcelona, Spain, 2008, pp. 165–168 (cit. on pp. 3, 17–19, 33, 44, 71).

[Cas+09] C. Castro-Herrera et al. "A recommender system for requirements elicitation in large-scale software projects." In: *Proceedings of the 2009 ACM Symposium on Applied Computing*. ACM. Hawaii, USA, 2009, pp. 1419–1426 (cit. on p. 52).

[Cha05] R. N. Charette. "Why software fails [software failure]." In: *Spectrum, IEEE* 42.9 (2005), pp. 42–49 (cit. on pp. 1, 2).

[Cha06] F. J. Chantree. "Identifying nocuous ambiguity in natural language requirements." PhD thesis. The Open University, 2006 (cit. on p. 48).

[Che+05] K. Chen et al. "An approach to constructing feature models based on requirements clustering." In: *13th IEEE Intl. Conf. on Req. Engineering (RE'05)*. Paris, France, 2005, pp. 31–40 (cit. on pp. 44, 71).

[Cia01] R. B. Cialdini. "The science of persuasion." In: *Scientific American* 284.2 (2001), pp. 76–81 (cit. on pp. 38, 39, 41, 42, 72, 75).

[Cle+09] J. Cleland-Huang et al. “Automated support for managing feature requests in open forums.” In: *Communications of the ACM* 52.10 (2009), pp. 68–74 (cit. on pp. 17, 20).

[CNR04] J. R. Curhan, M. A. Neale, and L. Ross. “Dynamic valuation: Preference changes in the context of face-to-face negotiation.” In: *Journal of Experimental Social Psychology* 40.2 (2004), pp. 142–151 (cit. on pp. 67, 82).

[CO90] F. Can and A. Ozkarahan. “Concepts and effectiveness of the clustering methodology for text databases.” In: *ACM Transactions on Database Systems* 15.4 (1990), pp. 483–517 (cit. on p. 24).

[CP12] L. Chen and P. Pu. “Critiquing-based recommenders: Survey and emerging trends.” In: *User Modeling and User-Adapted Interaction* 22.1-2 (2012), pp. 125–150 (cit. on p. 66).

[CR00] J. L. Cybulski and K. Reed. “Requirements classification and reuse: Crossing domain boundaries.” In: *Sixth International Conference on Software Reuse*. Lecture Notes in Computer Science. Vienna, Austria: Springer, 2000, pp. 190–210 (cit. on pp. 6, 49, 51).

[Dag+02] J. N. och Dag et al. “A feasibility study of automated natural language requirements analysis in market-driven development.” In: *Requirements Engineering* 7.1 (2002), pp. 20–33 (cit. on pp. 49, 58, 61).

[Dav03] A. Davis. “The art of requirements triage.” In: *IEEE Computer* 36.3 (2003), pp. 42–49 (cit. on pp. 2, 21, 73, 75, 83, 95).

[Dua+09] C. Duan et al. “Towards automated requirements prioritization and triage.” In: *Requirements Engineering* 14.2 (2009), pp. 73–89 (cit. on pp. 17, 21).

[Dum+11] H. Dumitru et al. “On-demand feature recommendations derived from mining public product descriptions.” In: *33rd ACM/IEEE International Conference on Software Engineering*. Waikiki, Honolulu, Hawaii: ACM/IEEE, 2011, pp. 181–190 (cit. on pp. 17, 19, 94).

[EL02] A. Eberlein and J. Leite. “Agile requirements definition: A view from requirements engineering.” In: *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE’02)*. Essen, Germany, 2002, pp. 4–8 (cit. on p. 48).

[FB08] A. Felfernig and R. Burke. “Constraint-based recommender systems: Technologies and research issues.” In: *ACM Intl. Conference on Electronic Commerce (ICEC’08)*. Innsbruck, Austria, 2008, pp. 17–26 (cit. on pp. 16, 23, 27, 85, 89, 95).

[FCM05] A. Felfernig, L. Chen, and M. Mandl. "RecSys'11 workshop on human decision making in recommender systems." In: *ACM Recommender Systems 2011 Workshop on Human Decision Making in Recommender Systems (Decisions@RecSys'11)*. Chicago, IL, 2005, pp. 389–390 (cit. on p. 30).

[Fel+07] A. Felfernig et al. "Persuasive recommendation: Serial position effects in knowledge-based recommender systems." In: *Persuasive technology*. Springer, 2007, pp. 283–294 (cit. on pp. 7, 67).

[Fel+09] A. Felfernig et al. "Plausible repairs for inconsistent requirements." In: *21st Intl. Joint Conference on Artificical Intelligence (IJCAI'09)*. Pasadena, CA, USA, 2009, pp. 791–796 (cit. on pp. 22, 28).

[Fel+10a] A. Felfernig et al. "Diagnosing inconsistent requirements preferences in distributed software projects." In: *3rd International Workshop on Social Software Engineering*. Paderborn, Germany, 2010, pp. 1–8 (cit. on pp. 17, 20, 22, 28, 29, 95).

[Fel+10b] A. Felfernig et al. "Recommendation and decision technologies for requirements engineering." In: *ICSE 2010 Workshop on Recommender Systems in Software Engineering*. Cape Town, South Africa, 2010, pp. 1–5 (cit. on pp. 15, 38, 83).

[Fel+12] A. Felfernig et al. "Group decision support for requirements negotiation." In: *Advances in User Modeling*. Ed. by Liliana Ardissono and Tsvi Kuflik. Vol. 7138. Springer Berlin / Heidelberg, 2012, pp. 105–116 (cit. on pp. 1, 9, 17, 22, 30, 33, 82, 85, 87, 91, 92).

[Fel+13] A. Felfernig et al. "An overview of recommender systems in requirements engineering." In: *Managing Requirements Knowledge Book*. Berlin Heidelberg: Springer, 2013, pp. 315–332 (cit. on p. 15).

[Fer+14] A. Ferrari et al. "Pragmatic ambiguity detection in natural language requirements." In: *1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. Karlskrona, Sweden: IEEE, 2014, pp. 1–8 (cit. on p. 5).

[Fes57] L. Festinger. *A theory of cognitive dissonance*. Vol. 1. Stanford University Press, 1957 (cit. on pp. 7, 67).

[Fir04] D. Firesmith. "Prioritizing requirements." In: *Journal of Object Technology* 3.8 (2004), pp. 35–48 (cit. on pp. 2, 7, 65).

[Fir05] D. Firesmith. "Are your requirements complete?" In: *Journal of Object Technology* 4.1 (2005), pp. 27–44 (cit. on p. 5).

[FLF11] C. Fitzgerald, E. Letier, and A. Finkelstein. “Early failure prediction in feature request management systems.” In: *19th IEEE International Conference on Requirements Engineering*. Trento, Italy: IEEE, 2011, pp. 229–238 (cit. on pp. 17, 20).

[FN12] A. Felfernig and G. Ninaus. “Group recommendation algorithms for requirements prioritization.” In: *Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. IEEE. Zurich, Switzerland, 2012, pp. 59–62 (cit. on pp. 2, 65, 66, 87).

[FNR12] A. Felfernig, G. Ninaus, and F. Reinfrank. “Eliciting stakeholder preferences for requirements prioritization.” In: *Decisions@ RecSys*. Dublin, Ireland, 2012, pp. 27–31 (cit. on p. 65).

[FS05] A. Fantechi and E. Spinicci. “A content analysis technique for inconsistency detection in software requirements documents.” In: *Requirements Engineering Workshop (WER2005)*. Porto, Portugal, 2005, pp. 245–256 (cit. on pp. 21, 31, 64, 95).

[FSR13] A. Felfernig, M. Schubert, and S. Reiterer. “Personalized diagnosis for over-constrained problems.” In: *23rd International Conference on Artificial Intelligence (IJCAI 2013)*. Peking, China, 2013, pp. 1990–1996 (cit. on p. 89).

[Gar11] Gartner Group. *Hype cycle for application development: Requirements elicitation and simulation*. 2011 (cit. on pp. 1, 47).

[GB01] P. Grünbacher and R. O. Briggs. “Surfacing tacit knowledge in requirements negotiation: Experiences using EasyWinWin.” In: *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. IEEE. Maui, HI, USA, 2001, pp. 1–8 (cit. on pp. 5, 98).

[Ger00] V. Gervasi. “Environment support for requirements writing and analysis.” PhD thesis. Pisa, Italy: University of Pisa, 2000 (cit. on p. 48).

[GKB08] A. Goknil, I. Kurtev, and K. van den Berg. “A metamodeling approach for reasoning about requirements.” In: *4th European Conference Model Driven Architecture - Foundations and Applications (ECMDA-FA)*. Vol. 5095. Lecture Notes in Computer Science. Berlin, Germany, 2008 (cit. on p. 48).

[GN02] V. Gervasi and B. Nuseibeh. “Lightweight validation of natural language requirements.” In: *Software: Practice and Experience* 32.2 (2002), pp. 113–133 (cit. on p. 48).

[Gol09] J. Golbeck. *Computing with social trust*. Springer, 2009 (cit. on pp. 16, 18, 23).

[Gon+98] J. Gonzalo et al. “Indexing with WordNet synsets can improve text retrieval.” In: *ACL/COLING Workshop on Usage of WordNet for Natural Language Processing*. Montréal, Canada, 1998 (cit. on p. 50).

[Grü00] P. Grünbacher. "Collaborative requirements negotiation with EasyWinWin." In: *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*. IEEE. Greenwich, London, 2000, pp. 954–958 (cit. on p. 67).

[Gru93] T. R. Gruber. "A translation approach to portable ontology specifications." In: *Knowledge acquisition* 5.2 (1993), pp. 199–220 (cit. on p. 107).

[GS03] T. Greitemeyer and S. Schulz-Hardt. "Preference-consistent evaluation of information in the hidden profile paradigm: Beyond group-level explanations for the dominance of shared information in group decisions." In: *Journal of Personality and Social Psychology* 84.2 (2003), pp. 322–339 (cit. on pp. 38, 39, 41, 43, 66, 70, 73).

[GZ05] V. Gervasi and D. Zowghi. "Reasoning about inconsistencies in natural language requirements." In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 14.3 (2005), pp. 277–330 (cit. on pp. 47, 48).

[Her+04] J. Herlocker et al. "Evaluating collaborative filtering recommender systems." In: *ACM Transactions on Information Systems* 22.1 (2004), pp. 5–53 (cit. on pp. 15, 85).

[HGR10] W. Heider, P. Gruenbacher, and R. Rabiser. "Negotiation constellations in reactive product line evolution." In: *4th International Workshop on Software Product Management (IWSPM'10)*. Sydney, Australia, 2010, pp. 63–66 (cit. on pp. 44, 70).

[HK05] R. Hastie and R. Kameda. "The robust beauty of majority rules in group decisions." In: *Psychological Review* 112.2 (2005), pp. 494–508 (cit. on p. 40).

[HL01] H. F. Hofmann and F. Lehner. "Requirements engineering as a success factor in software projects." In: *IEEE software* 18.4 (2001), p. 58 (cit. on pp. 2, 7, 33, 47, 65, 83).

[HM08] H. J. Happel and W. Maalej. "Potentials and challenges of recommendation systems for software development." In: *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. ACM. Atlanta, Georgia, USA, 2008, pp. 11–15 (cit. on p. 31).

[HRL12] S. W. Hansen, W. N. Robinson, and K. J. Lyytinen. "Computing requirements: Cognitive approaches to distributed requirements engineering." In: *45th Hawaii International Conference on System Science (HICSS)*. IEEE. Maui, HI, USA, 2012, pp. 5224–5233 (cit. on p. 47).

[HSS03] A. Hotho, S. Staab, and G. Stumme. "Ontologies improves text document clustering." In: *Proc. of the SIGIR 2003 Semantic Web Workshop*. Toronto, Canada, 2003, pp. 541–544 (cit. on pp. 50, 51).

[IPW10] P. G. Ipeirotis, F. Provost, and J. Wang. “Quality management on amazon mechanical turk.” In: *Proceedings of the ACM SIGKDD workshop on Human Computation*. ACM. Washington, USA, 2010, pp. 64–67 (cit. on p. 64).

[IR04] J. Iyer and D. Richards. “Evaluation framework for tools that manage requirements inconsistency.” In: *Proceedings of the 9th Australian Workshop on Requirements Engineering (AWRE’04)*. Adelaide, South Australia, 2004 (cit. on pp. 20, 94).

[Jan+10] D. Jannach et al. *Recommender systems: An introduction*. Cambridge University Press, 2010 (cit. on pp. 58, 83, 86).

[JBK04] 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*. Gallipoli, Italy, 2004, pp. 48–54 (cit. on pp. 3, 23, 26, 33, 37–39, 43, 70, 85, 87).

[Jun04] U. Junker. “QuickXplain: Preferred explanations and relaxations for over-constrained problems.” In: *19th National Conference on Artificial Intelligence (AAAI04)*. San Jose, CA, USA, 2004, pp. 167–172 (cit. on p. 29).

[Kon+97] J. Konstan et al. “GroupLens: Applying collaborative filtering to usenet news full text.” In: *Communications of the ACM* 40.3 (1997), pp. 77–87 (cit. on pp. 16, 19, 23–25).

[KS06] H. Kaiya and M. Saeki. “Using domain ontology as domain knowledge for requirements elicitation.” In: *14th IEEE International Conference Requirements Engineering*. IEEE. Minneapolis/St.Paul, Minnesota, USA, 2006, pp. 189–198 (cit. on pp. 51, 105, 107).

[Lam00] A. Van Lamsweerde. “Requirements engineering in the year 00: A research perspective.” In: *Proceedings of the 22nd international conference on Software engineering*. ACM. Limerick, Ireland, 2000, pp. 5–19 (cit. on pp. 47, 48).

[Lef97] D. Leffingwell. “Calculating the return on investment from more effective requirements management.” In: *American Programmer* 10.4 (1997), pp. 13–16 (cit. on p. 83).

[LF12] S. Lim and A. Finkelstein. “StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation.” In: *Software Engineering, IEEE Transactions on* 38.3 (2012), pp. 707–735 (cit. on pp. 17, 19).

[Lin99] K. R. Linberg. “Software developer perceptions about software project failure: A case study.” In: *Journal of Systems and Software* 49.2 (1999), pp. 177–192 (cit. on p. 1).

[LKT01] E. A. Lind, L. Kray, and L. Thompson. “Primacy effects in justice judgments: Testing predictions from fairness heuristic theory.” In: *Organizational behavior and human decision processes* 85.2 (2001), pp. 189–210 (cit. on p. 67).

[LMP04] M. Luisa, F. Mariangela, and N. I. Pierluigi. “Market research for requirements analysis using linguistic tools.” In: *Requirements Engineering* 9.1 (2004), pp. 40–56 (cit. on pp. 2, 6, 48).

[LQF10] S. Lim, D. Quercia, and A. Finkelstein. “Stakenet: Using social networks to analyse the stakeholders of large-scale software projects.” In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. Cape Town, South Africa: ACM/IEEE, 2010, pp. 295–304 (cit. on pp. 17–19, 83, 94).

[LRA08] 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*. Munich, Germany, 2008, pp. 22–28 (cit. on p. 28).

[LS04] H. Liu and P. Singh. “ConceptNet—A practical commonsense reasoning toolkit.” In: *BT technology journal* 22.4 (2004), pp. 211–226 (cit. on pp. 8, 97).

[LS06] S. Lichtenstein and P. Slovic. *The construction of preference*. Cambridge University Press, 2006 (cit. on p. 22).

[LSY03] G. Linden, B. Smith, and J. York. “Amazon.com recommendations: Item-to-item collaborative filtering.” In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80 (cit. on pp. 15, 85).

[LW00] D. Leffingwell and D. Widrig. *Managing software requirements: A unified approach*. Addison-Wesley Professional, 2000 (cit. on p. 47).

[Mas04] J. Masthoff. “Group modeling: Selecting a sequence of television items to suit a group of viewers.” In: *UMUAI* 14.1 (2004), pp. 37–85 (cit. on pp. 16, 26).

[Mas11] J. Masthoff. “Group recommender systems: Combining individual models.” In: *Recommender Systems Handbook*. Springer, 2011, pp. 677–702 (cit. on pp. 3, 22, 33, 37–40, 43, 66, 70, 76, 81, 85, 87, 89, 114).

[MC11] B. Mobasher and J. Cleland-Huang. “Recommender systems in requirements engineering.” In: *AI Magazine* 32.3 (2011), pp. 81–89 (cit. on pp. 15, 17–20, 83).

[McF99] D. McFadden. “Rationality for economists?” In: *Journal of Risk and Uncertainty* 19.1 (1999), pp. 73–105 (cit. on pp. 4, 22, 33, 44, 71).

[MG10] C. M. Meyer and I. Gurevych. “Worth its weight in gold or yet another resource—A comparative study of Wiktionary, OpenThesaurus and GermaNet.” In: *Computational Linguistics and Intelligent Text Processing*. Springer, 2010, pp. 38–49 (cit. on pp. 50, 53, 54).

[MKD07] S. Marczak, I. Kwan, and D. Damian. “Social networks in the study of collaboration in global software teams.” In: *International Conference on Global Software Engineering (ICGSE'07)*. Munich, Germany, 2007 (cit. on p. 18).

[MKF14] W. Maalej, Z. Kurtanovic, and A. Felfernig. “What stakeholders need to know about requirements.” In: *4th International Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE. Karlskrona, Sweden, 2014, pp. 64–71 (cit. on p. 2).

[MMW06] D. Milne, O. Medelyan, and I. H. Witten. “Mining domain-specific thesauri from Wikipedia: A case study.” In: *Proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence*. IEEE Computer Society. Hong Kong, China, 2006, pp. 442–448 (cit. on p. 51).

[MR00] R. Mooney and L. Roy. “Content-based book recommending using learning for text categorization.” In: *Proceedings of the fifth ACM conference on Digital libraries*. ACM. San Antonio, TX, USA, 2000, pp. 195–204 (cit. on pp. 23, 85).

[MR01] S. Mohammed and E. Ringseis. “Cognitive diversity and consensus in group decision making: The role of inputs, processes, and outcomes.” In: *Organizational behavior and human decision processes* 85.2 (2001), pp. 310–335 (cit. on p. 82).

[MS10] A. Mojzisch and S. Schulz-Hardt. “Knowing other's preferences degrades the quality of group decisions.” In: *Journal of Personality & Social Psychology* 98.5 (2010), pp. 794–808 (cit. on pp. 38, 39, 41, 43, 70, 73).

[MT09] W. Maalej and A. Thurimella. “Towards a research agenda for recommendation systems in requirements engineering.” In: *International Workshop on Managing Requirements Knowledge*. Atlanta, USA, 2009 (cit. on pp. 15, 44, 71, 83).

[Nab05] D. Naber. “OpenThesaurus: ein offenes deutsches Wortnetz.” In: *Sprachtechnologie, mobile Kommunikation und linguistische Ressourcen: Beiträge zur GLDV-Tagung, Bonn, Germany* (2005), pp. 422–433 (cit. on p. 53).

[NE00] B. Nuseibeh and S. Easterbrook. “Requirements engineering: A roadmap.” In: *Proceedings of the Conference on the Future of Software Engineering*. ACM. Limerick, Ireland, 2000, pp. 35–46 (cit. on pp. 38, 47, 49, 53).

[NFR12] G. Ninaus, A. Felfernig, and F. Reinfrank. “Anonymous preference elicitation for requirements prioritization.” In: *Foundations of Intelligent Systems, 20th International Symposium, ISMIS 2012*. Vol. 7661. Macau, China: Springer, 2012, pp. 349–356 (cit. on pp. 11, 65).

[Nin+14a] G. Ninaus et al. “Content-based recommendation techniques for requirements engineering.” In: *1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. Karlskrona, Sweden: IEEE, 2014, pp. 27–34 (cit. on pp. 3, 11, 47).

[Nin+14b] G. Ninaus et al. “INTELLIREQ: Intelligent techniques for software requirements engineering.” In: *21st European Conference on Artificial Intelligence / Prestigious Applications of Intelligent Systems (PAIS 2014)*. Prague, Czech Republic, 2014, pp. 1161–1166 (cit. on pp. 2, 9, 10, 51, 83).

[Nin12] G. Ninaus. “Using group recommendation heuristics for the prioritization of requirements.” In: *Proceedings of the sixth ACM conference on Recommender systems*. ACM. Dublin, Ireland, 2012, pp. 329–332 (cit. on pp. 2, 12, 65, 87).

[PB97] M. Pazzani and D. Billsus. “Learning and revising user profiles: The identification of interesting web sites.” In: *Machine Learning* 27.3 (1997), pp. 313–331 (cit. on pp. 15, 16, 18, 23, 85).

[PC08] P. Pu and L. Chen. “User-involved preference elicitation for product search and recommender systems.” In: *AI Magazine* 29.4 (2008), pp. 93–103 (cit. on pp. 4, 34, 39, 44, 71).

[Pei+10] B. Peischl et al. “Constraint-based recommendation for software project effort estimation.” In: *Journal of Emerging Technologies in Web Intelligence* 2.4 (2010), pp. 282–290 (cit. on p. 94).

[PH97] A. Pinsonneault and N. Heppel. “Anonymity in group support systems research: A new conceptualization, measure, and contingency framework.” In: *Journal of Management Information Systems* 14.3 (1997), pp. 89–108 (cit. on pp. 65, 69).

[PJ90] J. K. Pinto and S. J. Mantel Jr. “The causes of project failure.” In: *Engineering Management, IEEE Transactions on* 37.4 (1990), pp. 269–276 (cit. on p. 1).

[PL02] P. Pantel and D. Lin. “Discovering word senses from text.” In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. Edmonton, AB, Canada, 2002, pp. 613–619 (cit. on p. 51).

[Poh13] K. Pohl. “The three dimensions of requirements engineering.” In: *Seminal Contributions to Information Systems Engineering*. Springer, 2013, pp. 63–80 (cit. on p. 5).

[Poh96] K. Pohl. *Process-centered requirements engineering*. John Wiley & Sons, Inc., 1996 (cit. on p. 33).

[QB11] A. J. Quinn and B. B. Bederson. "Human computation: A survey and taxonomy of a growing field." In: *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM. Vancouver, Canada, 2011, pp. 1403–1412 (cit. on p. 97).

[Reg+01] B. Regnell et al. "Requirements mean decisions! - Research issues for understanding and supporting decision-making in requirements engineering." In: *1st Swedish Conference on Software Engineering Research and Practice (SERP 01)*. Ronneby, Sweden, 2001, pp. 49–52 (cit. on pp. 3–5).

[Rei87] R. Reiter. "A theory of diagnosis from first principles." In: *Artificial intelligence* 32.1 (1987), pp. 57–95 (cit. on pp. 20, 29, 89, 94).

[Ren+13] D. Renzel et al. "Requirements Bazar: Social requirements engineering for community-driven innovation." In: *RE 2013*. Rio de Janeiro, Brazil, 2013, pp. 326–327 (cit. on p. 83).

[REP03] G. Ruhe, A. Eberlein, and D. Pfahl. "Trade-off analysis for requirements selection." In: *Journal of Software Engineering and Knowledge Engineering (IJSEKE)* 13.4 (2003), pp. 354–366 (cit. on pp. 21, 22).

[RS05] G. Ruhe and M. O. Saliu. "The art and science of software release planning." In: *IEEE Software* 22.6 (2005), pp. 47–53 (cit. on p. 22).

[RSW08] B. Regnell, R. B. Svensson, and K. Wnuk. "Can we beat the complexity of very large-scale requirements engineering?" In: *Requirements Engineering: Foundation for Software Quality*. Springer, 2008, pp. 123–128 (cit. on p. 2).

[RWZ10] M. Robillard, R. Walker, and T. Zimmermann. "Recommendation systems for software engineering." In: *IEEE Software* 27.4 (2010), pp. 80–86 (cit. on p. 94).

[Rya93] K. Ryan. "The role of natural language in requirements engineering." In: *Proceedings of IEEE International Symposium on Requirements Engineering*. IEEE. San Diego, CA, USA, 1993, pp. 240–242 (cit. on pp. 2, 48).

[Sch98] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., 1998 (cit. on p. 22).

[SM99] S. Scott and S. Matwin. "Feature engineering for text classification." In: *Proceedings of the Sixteenth International Conference on Machine Learning*. Vol. 99. Citeseer. Bled, Slovenia, 1999, pp. 379–388 (cit. on p. 50).

[Som11] I. Sommerville. *Software engineering*. Pearson, 2011 (cit. on pp. 15, 83).

[SSK00] D. Schkade, C. R. Sunstein, and D. Kahneman. “Deliberating about Dollars: The severity shift.” In: *Columbia Law Review* (2000), pp. 1139–1175 (cit. on pp. 79, 81).

[Tab+11] M. Taboada et al. “Lexicon-based methods for sentiment analysis.” In: *Computational linguistics* 37.2 (2011), pp. 267–307 (cit. on p. 51).

[Tsa93] E. Tsang. *Foundations of constraint satisfaction*. London: Academic Press, 1993 (cit. on pp. 20, 90).

[Tsu+05] M. Tsunoda et al. “Javawock: A Java class recommender system based on collaborative filtering.” In: *SEKE 2005*. Taipei, Taiwan, 2005, pp. 491–497 (cit. on p. 94).

[Voo93] E. M. Voorhees. “Using WordNet to disambiguate word senses for text retrieval.” In: *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. Pittsburgh, PA, USA, 1993, pp. 171–180 (cit. on p. 50).

[WE86] D. Von Winterfeldt and W. Edwards. *Decision analysis and behavioral research*. Vol. 604. Cambridge University Press Cambridge, 1986 (cit. on p. 66).

[WF05] I. H. Witten and E. Frank. *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005 (cit. on pp. 16, 23, 24, 31, 64, 95).

[Wie99] K. Wiegers. “First things first: Prioritizing requirements.” In: *Software Development* 7.9 (1999), pp. 48–53 (cit. on pp. 7, 65, 66).

[WK04] L. Wallace and M. Keil. “Software project risks and their effect on outcomes.” In: *Communications of the ACM* 47.4 (2004), pp. 68–73 (cit. on p. 1).

[Yan+08] D. Yang et al. “WikiWinWin: A Wiki based system for collaborative requirements negotiation.” In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*. IEEE. Waikoloa, Big Island, Hawaii, 2008, p. 24 (cit. on p. 2).

[Zav97] P. Zave. “Classification of research efforts in requirements engineering.” In: *ACM Computing Surveys (CSUR)* 29.4 (1997), pp. 315–321 (cit. on p. 1).

[ZCW92] J. A. Zuber, H. W. Crott, and J. Werner. “Choice shift and group polarization: An analysis of the status of arguments and social decision schemes.” In: *Journal of Personality and Social Psychology* 62.1 (1992), p. 50 (cit. on pp. 7, 79).