

Dissertation

**Intelligent Recommendation Techniques
for Requirements Engineering**

Dipl.-Ing. Ralph Samer, BSc

Graz, February 2021

*Institute of Software Technology
Graz University of Technology*



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

Abstract (English)

Software systems have become an integral part of our daily life. The rising complexity as well as the increasing extent of software systems, which are both driven by a continuously growing number of requirements, have a direct impact on the planning and development of these systems. During the design and development process of software systems, the *requirements engineering* (RE) phase usually plays a decisive role. Precise and careful RE represents a major challenge for stakeholders in complex software projects. Inefficient RE can lead to a high expenditure of time and tremendous follow-up costs. In many projects, the primary cause of overspending can be attributed to mismanagement of RE. In the worst case, a missing or an insufficient consideration of RE could even result in project failure. In order to satisfy the needs of successful RE, technical solutions are required that support stakeholders to make correct decisions in RE tasks. For this reason, the main focus of this thesis lies on the application of intelligent recommendation technologies in RE. In this thesis, we shed light on some important RE tasks which are predestined for the application of recommendation-based solutions. This includes the *prioritization of requirements*, the *distribution and assignment of requirements to stakeholders*, and the *identification of dependency relationships between requirements*. Within the scope of the European research project OPENREQ, we developed and evaluated innovative recommendation approaches to support decision-making in the aforementioned tasks. The technologies presented in this thesis include (a) *innovative user interfaces for group-based prioritizations of requirements*, (b) *approaches based on machine learning to recommend requirements to developers* as well as to (c) *identify dependency relationships between requirements*, and (d) *group recommendation services for assigning requirements to stakeholders for industrial environments*. Our recommendation approaches, including the evaluation results, represent a fundamental part of our research work and are presented in this thesis. These approaches aim to improve the decision quality and the efficiency of the RE process. The improvements are achieved through the use of optimized mechanisms to enhance the quality of RE-related decisions, which differ significantly from existing solutions. These mechanisms advance the state-of-the-art and include concepts that lead to more exchange of contrary information between stakeholders in group decision-making processes, approaches to counteract *cold-start issues*, and specific recommendation solutions for open-source communities.

Abstract (German)

Softwaresysteme sind heutzutage ein integraler Bestandteil unseres täglichen Lebens. Die zunehmende Komplexität und der größer werdende Umfang, der von einer stetig wachsenden Anzahl an Anforderungen getrieben ist, hat direkte Auswirkungen auf die Planung und Entwicklung von Softwaresystemen. Bei der Entwicklung von Softwaresystemen spielt vor allem die Phase des *Requirements Engineering* (Anforderungsanalyse) eine entscheidende Rolle. Dabei stellt die sorgfältige Abwicklung von Requirements Engineering eine große Herausforderung bei der Entwicklung von komplexen Softwareprojekten dar. Ineffizientes Requirements Engineering kann zu einem hohen zeitlichen Mehraufwand und erheblichen Folgekosten führen. Bei vielen Projekten ist die Ursache überschreitender Budgets darin zu suchen, dass Requirements Engineering von Anfang an nicht ordentlich durchgeführt wurde. Im schlimmsten Fall drohen bei einer mangelhaften oder fehlenden Berücksichtigung von Requirements Engineering sogar Projektabbrüche. Um den Requirements Engineering Prozess bestmöglich zu begleiten, bieten sich technische Lösungen an, die Stakeholder bei kritischen Entscheidungen in Requirements Engineering Aufgaben unterstützen. Aus diesem Grund befasst sich diese Abschlussarbeit mit intelligenten Empfehlungstechnologien im Bereich Requirements Engineering. Im Rahmen dieser Arbeit beschäftigen wir uns mit einigen wichtigen Arbeitsbereichen der Anforderungsanalyse, die für den Einsatz empfehlungsbasierter Lösungen prädestiniert sind. Zu den wichtigsten Schritten zählen die *Priorisierung von Anforderungen*, die *Verteilung der Anforderungen an die Stakeholder* und die *Feststellung von Abhängigkeitsbeziehungen zwischen Anforderungen*. Im Rahmen des europäischen Forschungsprojekts OPENREQ wurden verschiedene intelligente Empfehlungssysteme für diese Bereiche entwickelt und evaluiert. Die dabei entwickelten Empfehlungslösungen umfassen (a) *innovative Benutzeroberflächen zur gruppenbasierten Evaluierung und Priorisierung von Anforderungen*, (b) *Empfehlungsansätze basierend auf maschinellem Lernen zur Vorhersage von Anforderungen* sowie zur (c) *Erkennung von Abhängigkeitsbeziehungen zwischen Anforderungen* und (d) *Gruppenempfehlungsdienste zur Zuweisung von Anforderungen* im industriellen Umfeld. Die vorgestellten Ansätze sowie die Ergebnisse der Evaluierungen sind ein wesentlicher Bestandteil unserer Forschungsarbeit und werden in dieser Abschlussarbeit präsentiert. Die Empfehlungsansätze zielen darauf ab, die Qualität von kritischen Entscheidungen in Requirements Engineering Abläufen zu verbessern, um effizientere Entscheidungsprozesse zu erhalten. Unsere Empfehlungsansätze wenden optimierte Mechanismen zur Verbesserung der Entscheidungsqualität

an, die sich von bestehenden Lösungen maßgeblich unterscheiden. Dazu zählen sowohl Konzepte, die zu kritischerem Informationsaustausch bei Gruppenentscheidungen führen, als auch verbesserte Ansätze, die *Cold-Start Problemen* entgegenwirken und sich an Open-Source-Entwickler richten.

Acknowledgement

First and foremost, I want to thank my supervisor *Univ.-Prof. Dr.techn. Dipl.-Ing. Alexander Felfer-nig* for the continuous support of my PhD study. He has always challenged my mind with new and innovative ideas and has provided me comprehensive guidance with his broad expertise and knowledge throughout my time at the institute. Our team meetings and conversations were helpful and vital in inspiring me to work even more precisely and professionally.

Moreover, I also want to express my sincerest thanks to my parents (*Gerhard* and *Brigitte*) and my grandmother *Maria*, who have supported me throughout my whole studies. Special thanks also go to my girlfriend *Nilobol* for her patience and encouragement as well as to all other family members, my friends and colleagues (*Martin, Georg, Müslüm, Patrick, Wolfgang, and Michael*) for their essential and unconditional support during my studies.

At this point, I also want to thank the *European Union* for the financial support of the OPENREQ research project (No. 732463, Horizon 2020 program). Finally, I want to recall and thank all our project partners from Germany (the company *Vogella GmbH* and the *University of Hamburg*), Spain (*Polytechnic University of Catalonia*), Austria (the company *Siemens*), Finland (the research teams at the *Qt company* and the *University of Helsinki*), and Italy (the companies *Wind Tre* and *Engineering Ingegneria Informatica S.p.A.*) for their research contributions, the insightful discussions, and our successful research cooperation.

Ralph Samer
Graz, 2021

AFFIDAVIT

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Graz, 2021-02-19

Place, Date



Signature

EIDESSTATTLICHE ERKLÄRUNG

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. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Graz, am 19.02.2021

Ort, Datum



Unterschrift

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Research Objectives	6
1.3. Contributions	10
1.4. Thesis Outline	15
2. Choice Scenarios Beyond Basic Recommendation	19
2.1. Abstract	19
2.2. Introduction	19
2.3. Ranking	23
2.4. Packaging	24
2.5. Parametrization	24
2.6. Configuration	25
2.7. Release Planning	26
2.8. Triage	27
2.9. Resource Balancing	28
2.10. Sequencing	28
2.11. Polls and Questionnaires	30
2.12. Voting	31
2.13. Further Aspects of Choice Scenarios	31
2.14. Conclusions and Research Issues	33
3. Recommender Systems in Requirements Engineering	35
3.1. Abstract	35
3.2. Introduction	35
3.3. Research Methodology	38
3.4. Related Work	40

3.5.	Recommendation Technologies in Requirements Engineering	42
3.5.1.	Basic Recommendation Algorithms in Requirements Engineering	43
3.5.2.	Advanced Recommendation Algorithms in Requirements Engineering	48
3.6.	Application Areas of Recommenders in Requirements Engineering	58
3.6.1.	Recommenders in Traditional Requirements Engineering	58
3.6.2.	Requirements Management Platforms	71
3.7.	Selection of Recommendation Algorithms	80
3.8.	Open Research Topics	84
3.9.	Conclusion	86
4.	New Approaches to the Identification of Dependencies between Requirements	89
4.1.	Abstract	89
4.2.	Introduction	89
4.3.	User Study & Dataset	91
4.4.	Preprocessing & Feature Extraction	92
4.4.1.	Extraction of TF-IDF Features	92
4.4.2.	Extraction of Probabilistic Features	93
4.5.	Approach	94
4.5.1.	Classification (Approach I)	94
4.5.2.	Latent Semantic Analysis (Approach II)	95
4.6.	Evaluation & Discussion	96
4.7.	Conclusion & Future Work	98
5.	Group Recommender User Interfaces for Improving Requirements Prioritization	101
5.1.	Abstract	101
5.2.	Introduction	101
5.3.	Group Recommendation for Requirements Prioritization	104
5.3.1.	One-dimensional Rating Approach	104
5.3.2.	Multi-attribute Utility Rating Approach	104
5.3.3.	Argumentation-based Rating Approach	109
5.4.	Evaluation	110
5.4.1.	Experimental Setup	111
5.4.2.	Results & Discussion	112
5.5.	Threats to Validity	117
5.5.1.	Internal Validity	117
5.5.2.	External Validity	117
5.6.	Future Work	118
5.7.	Conclusions	118

6. Group Decision Support for Requirements Management Processes	119
6.1. Abstract	119
6.2. Introduction	119
6.3. Application Scenario	121
6.3.1. Traditional RM Process	121
6.3.2. RM Process with Group Decision Support	124
6.4. Potential Issues of Group Decision Support	126
6.5. Group Decision Support for Bidding Processes	126
6.6. Conclusion and Future Work	129
7. Towards Utility-based Prioritization of Requirements in Open-Source Environments	131
7.1. Abstract	131
7.2. Introduction	131
7.3. Utility-based Prioritization	135
7.4. Utility-based Prioritization in BUGZILLA	137
7.5. Taking into Account Dependencies	139
7.6. Conclusion and Future Work	140
8. Towards Issue Recommendation for Open-Source Communities	143
8.1. Abstract	143
8.2. Introduction	143
8.3. Methodology	146
8.3.1. Datasets	146
8.3.2. Recommendation Approach	147
8.4. Evaluation & Discussion	151
8.5. Future Work	157
8.6. Conclusion	158
9. Intelligent Recommendation & Decision Technologies for Community-Driven RE	159
9.1. Abstract	159
9.2. Introduction	159
9.3. OPENREQ Recommendation Technologies	161
9.3.1. Requirements Elicitation	162
9.3.2. Requirement Dependency Detection	164
9.3.3. Prioritization and Evaluation of Requirements	165
9.3.4. Stakeholder Recommendation	166
9.3.5. Release Planning and Configuration	168
9.3.6. Quality Assurance	168
9.4. OPENREQ User Interface	169

Contents

9.5. User Studies and Benefits	172
9.6. Related and Future Work	175
9.7. Conclusion	176
10. Conclusions & Future Work	177
10.1. Conclusions	177
10.2. Future Work	184
A. Further Study Results of Chapter 5	189
B. Detailed Evaluation Results of Chapter 8	191
List of Figures	193
List of Tables	197
Bibliography	203

Introduction

1.1. Motivation

Nowadays, our life and work is highly influenced by software systems. Sometimes, software systems are an integral part of a device which we use on a daily basis, such as a mobile phone. However, software-based functionality often seems to be hidden, like in a car, where only the experience of a necessary software update at the service station makes us aware that these technical machines can no longer work without software. Thus, software systems have become indispensable. The central role of software and its importance is also reflected in various statistics. For example, Gartner Research (2020) reports that the global *information technology* spending on enterprise software has increased by around 94% in the last 10 years (from 2010 to 2019).

Requirements engineering represents a key factor for the success of a software project. It is a complex engineering field that involves the collection, documentation, coordination, analysis, and management of requirements and their changes throughout the lifecycle of a product (Sommerville, 2010; Hoffmann et al., 2004; Cant et al., 2006). In general, requirements describe what a customer can expect from a technical product (e.g., a software product) in terms of properties, benefits, conditions, and goals (Ebert, 2014). A requirement can either be (1) a *functional* requirement which is related to a technical feature (such as, "the camera must have an integrated timer to automatically capture a picture"), a use case, or any other piece of functionality that should be part of the developed product, or (2) a *non-functional* requirement which relates to aspects such as correctness, reliability, usability, performance, profit, privacy, security, or technical safety.

Although requirements engineering is a discipline to organize any kind of technical projects, the focus of this thesis lies on software projects. However, the aspects, statements, and results presented in this work can also be regarded as generally valid for all kinds of systems engineering (e.g., aircraft engineering, aerospace engineering, railway engineering). Research in the field of requirements en-

engineering can be regarded as a continuously ongoing scientific undertaking (Fernández, 2018). From the perspective of a traditional software development model (such as the waterfall model), requirements engineering typically appears as the initial phase in a software project's development process and should be completed before the software product is designed and implemented. However, agile development methods (such as SCRUM (Darwish and Megahed, 2016)) - which have grown in popularity over the last decade - consider requirements engineering as an ongoing process that continues through the lifetime of a software project (this meets the aforementioned definition of requirements engineering more precisely). Requirements engineering represents a comprehensive discipline embedded in a process which consists of different core activities that can be overlapping, repetitive, and iterative (see Chapter 3). These activities are the *definition and elicitation of requirements*, the *analysis and negotiation of requirements*, *quality assurance*, and *release planning*. During these activities, requirements are defined, analyzed, managed, and prioritized.

Nowadays, modern software projects typically consist of a high number of requirements which can range up to many hundreds or thousands of requirements. This introduces high complexity to the *requirements engineering process* which poses many challenges for stakeholders (stakeholders can be, e.g., customers, project managers, technical engineers, or software developers). Examples of such challenges include the *identification of relevant requirements* (requirements triage), the *assignment of suitable stakeholders* to evaluate and take over responsibility for requirements, the *identification of dependencies* between requirements (dependency detection), and *resource planning* (e.g., release planning).

In many software projects, the budget and the time to work on a project are limited. Therefore, efficient time management plays a vital role in most of these projects and it becomes imperative to avoid all possible sources of future errors in advance which can quickly lead to costly time overruns. Past studies reveal that around 60 percent of development errors originate from faulty requirements engineering (Boehm, 1981). According to Gartner Research (2014), *requirements represent the first source of defects for service projects and the third source of product defects – the costs to fix defects range from 70 USD at the requirements phase to 14,000 USD in the production phase*. Consequently, high effort investments in requirements engineering are crucial, in order to ensure that the project keeps within budget and time expectations. Davis (2005) demonstrates that around 40 percent of project failures are due to missing or faulty requirements engineering. Although there exists a variety of different definitions of *project failure* in the literature (Pinto and Mantel, 1990), common conventions of *project failure* include practical scenarios where a project must be cancelled ("project failure"), an on-time delivery of the software product can not be guaranteed ("schedule overrun"), the running costs of the project have exceeded the limits of the project budget ("budget overrun"), some requirements can not be satisfied ("wrong or missing functionality / requirements"). Examples of common issues that can negatively affect *project success* are *incomplete or unclear definitions*

of the requirements, missing or insufficient prioritizations of the requirements, and unconsidered or overlooked relationships (dependencies) between the requirements. In particular, the late discovery of such issues inevitably leads to significantly increased costs in terms of money and time. According to Cleland-Huang et al. (2003), the costs for requirement changes can vary from between 40 and 90 percent of the total development costs. This comes at the risk of exceeding the limited project budget that is available and can lead to project failure in the worst case. In order to ensure a proper completion of a software project, strong requirements engineering efforts are needed in early phases of the project.

However, even though companies are willing to focus on requirements engineering, the high complexity and large size of many software projects represents an overwhelming challenge for stakeholders (Fucci et al., 2018; Damasiotis et al., 2017). For example, a high level of software complexity quickly leads to a large number of requirements that have to be evaluated, and rapidly growing requirement documents. As a result, stakeholders can easily lose track of things and even standard tasks such as, for example, assigning suitable stakeholders to requirements manually, can become increasingly difficult. In addition, the differentiation between which paragraphs within a textual requirement document define a new requirement and which pieces of text of the document relate to a requirement becomes more and more difficult for stakeholders as the complexity of the software increases. Moreover, the manual detection of dependencies between requirements is also becoming increasingly difficult and the number of requirement pairs to analyze (in order to find dependent requirement-pairs) increases quadratically with the number of requirements.

These aforementioned challenges trigger the increasing need for intelligent support in requirements engineering. Specific academic interests for computer-aided support in requirements engineering can be traced back to the early eighties with the publication of Teichroew and Sayani (1980) proposing the use of *computer-aided requirements engineering* (CARE) to support requirements engineering tasks. These interests have increased significantly with the ongoing developments and advances in the area of *recommender systems* over the past 15 years. In this context, the *Netflix Prize challenge* (Bell et al., 2007) from 2006 can be mentioned, which was the first major scientific competition in the area of recommender systems. In general, recommender systems can be defined as *decision-support tools which find matching objects in large spaces of available objects or generate suitable objects as output* (Burke, 2002). More precisely, according to Ricci et al. (2010):

“Recommender systems are tools for interacting with large and complex information spaces. They provide a personalized view of such spaces, prioritizing items likely to be of interest to the user.”

Recommender systems can significantly accelerate the search process for users by proposing suitable selection options – this often results in a higher decision quality (Chen et al., 2013; Isinkaye et al., 2015; Pathak et al., 2010). The emergence of recommendation-based services led to an unprecedented

increase of their popularity and importance. This can be explained by the progress these systems have made over recent years, which has extended to numerous improvements and new recommendation approaches. Some examples were further developments in areas such as *matrix factorization* (Koren et al., 2009), *group-based recommenders* (Felfernig et al., 2018), or *deep learning* (Goodfellow et al., 2016). It is important to mention that recommendation approaches based on these new developments work particularly well as soon as they are fed with large amounts of data. In general, this applies to the area of requirements engineering, since software projects usually consist of large sets of requirements which results in large amounts of requirements data. Therefore, recommender systems are also suitable solutions to address investments in computer-assisted requirements engineering (Mobasher and Cleland-Huang, 2011; Felfernig et al., 2013). The use of recommender systems can help to improve the overall quality of requirements engineering processes (Palomares et al., 2018; Felfernig et al., 2013) by reducing risks such as cost and time overruns, as well as project failures. The reason behind this is that requirements engineering is a very decision-driven discipline and recommender systems are particularly helpful in complex decision scenarios. The high complexity of software leads to large amounts of requirement data, many involved stakeholders, and many alternatives that need to be analyzed and evaluated by the stakeholders before decision-making (Johann and Maalej, 2015; Palomares et al., 2018; Davis, 2003). Beyond that, a low quality of requirements also introduces further complexity to the decision-making process (e.g., via inconsistencies or incomplete definitions) (Palomares et al., 2018).

Recommender systems are suitable solutions to effectively tackle the aforementioned problems. They have great potential and their benefits are manifold. These systems can provide critical decision support for stakeholders in requirements engineering (Castro-Herrera et al., 2009; Mobasher and Cleland-Huang, 2011; Ninaus et al., 2014). Furthermore, recommenders can provide help in complex situations where stakeholders would otherwise not have sufficient knowledge for making high-quality decisions (Palomares et al., 2018). Thereby, recommenders can tailor recommendations to the specific preferences and information needs of stakeholders.

Recommenders can be applied to support many different requirements engineering tasks – starting from the collection of project requirements, through the assignment of stakeholders to requirements, and the prioritization of requirements. For example, *content-based filtering* recommendations play an important role in this context. The basic idea of content-based approaches (van Meteren and van Someren, 2000; Pazzani and Billsus, 2007) consists in recommending new items to users that are similar to the items the users have preferred in the past. Requirements are often written in natural language and can be characterized by additional attributes such as implementation effort, or the type of a requirement (e.g., technical feature, performance requirement, or financial requirement). This way, requirements can be seen as text documents with additional meta-data (the attributes). In such cases, content-based recommendation approaches can be used to recommend requirements by taking

into account the given textual information and attributes related to the requirements. Content-based recommenders can help to assist stakeholders, for example, by recommending requirements from past projects for reuse in new projects, by proposing suitable stakeholders for a requirement, or by recommending new requirement dependencies.

Another recommendation approach that can be applied in requirements engineering is *collaborative filtering*. Collaborative filtering is based on the concept of word-of-mouth promotions (Ekstrand et al., 2011; Goldberg et al., 1992). In practice, people often base their decisions on suggestions received from other individuals they know and trust. Collaborative filtering implements this concept by finding users who are similar to the current user and by recommending items to the current user that were preferred by these similar users (Ekstrand et al., 2011). Since collaborative filtering approaches do not take into account the content of the requirements (e.g., requirement text, requirement type, or attached content), they can be applied as soon as sufficient evaluation / rating data is available. One example of collaborative filtering in requirements engineering is to assist stakeholders in the identification of relevant requirements based on past requirement evaluation / rating data. Further application examples typically follow the idea of supporting stakeholders in the navigation within large collections of requirements or to suggest further interesting artifacts (such as use case diagrams, design documents, or business process diagrams) based on their taste.

In contrast to the aforementioned recommendation approaches, *knowledge-based recommender systems* represent a completely different recommendation approach. However, these systems are also of special interest for requirements engineering. Knowledge-based recommenders (Felfernig et al., 2014) focus on *high-involvement items* (or *complex items*), such as financial services, digital cameras, or tourist destinations. A knowledge-based algorithm determines recommendations based on predefined recommendation settings (including stakeholder preferences), as well as specific domain knowledge and meta-information about the item assortment. In the context of requirements engineering, knowledge-based recommenders can be used to generate and recommend release plans based on given constraints (such as release deadlines, requirement dependencies, resource limitations) or to find inconsistencies (e.g., contradictory requirement dependencies) in a requirements model.

Another relevant recommendation approach is *group recommender systems*. Group recommenders (Masthoff, 2015; Felfernig et al., 2018) aim at fostering decision-consensus among stakeholders where decisions are made in groups. A major benefit of these systems is that they can foster discussions. Group recommenders are based on the principle of aggregating preferences of individual group members (stakeholders) into a single recommendation for the group¹. In order to generate group recommendations, there are different aggregation functions that represent the upper layer of the group recommendation system. The predictions are generated by the aforementioned recommendation ap-

¹Note that there also exist group recommendation solutions which aggregate single-user recommendations of each group member (see Section 3.5.2).

proaches which represent the underlying layer of the system. In requirements engineering, there also exist many application scenarios for group recommenders. Some application examples (Boehm et al., 2001; Felfernig et al., 2011; Ninaus, 2012; Farshidi et al., 2018) address the evaluation and prioritization of requirements or the group-based elicitation of requirements.

1.2. Research Objectives

Within the scope of the European research project OPENREQ, we developed numerous recommendation tools with the goal of improving the quality of requirements engineering. OPENREQ!LIVE represents a CARE environment (see Section 1.1) that includes these recommendation tools. In contrast to existing recommender applications that focus on specific requirements engineering tasks, the OPENREQ project addresses the complete requirements engineering process as a whole (see Chapter 9). The following research objectives were identified for this thesis.

1. Support stakeholders in the identification of requirement dependencies

Many requirements engineering tasks are well supported by recommendation solutions. Examples thereof are the *automated identification and recommendation of paragraphs in requirements documents that represent requirements* (Winkler and Vogelsang, 2016; Abualhaija et al., 2019), the *suggestion of requirements from past projects to reuse in new projects* (Dumitru et al., 2011; Ivan et al., 2016), or *recommendations to improve quality assurance* (Fitzgerald et al., 2011; Ninaus et al., 2014). These approaches aim to improve the quality of requirements engineering by proactively assisting stakeholders. However, one important task that lacks intelligent decision support is the identification of dependency relationships between requirements. A dependency relationship expresses in which way requirements depend on each other. Dependencies are essential for the detection of redundancies and inconsistencies between the requirements (Aguilar et al., 2012). Furthermore, requirements and dependencies define the basis for release planning (Ruhe, 2010). There exist different types of requirement dependencies such as *requires*, *includes*, or *excludes*. Among these types, the type *requires* is the most frequently occurring dependency type (Ferber et al., 2002). Hence, special attention should be given to this dependency type. An early identification of dependencies is essential for a project, since the late discovery of these dependencies can lead to negative consequences such as increased costs or unfulfilled deadlines (Leffingwell, 1997; Mobasher and Cleland-Huang, 2011; Ruhe, 2010; Vogelsang and Fuhrmann, 2013) (see also Section 1.1). In most cases, requirements represent text that has been written in natural language (Berry and Kamsties, 2004; Ferrari et al., 2014). Due to the high complexity and size of software projects, the manual identification of requirement dependencies is very time-intensive (Vogelsang and Fuhrmann, 2013; Deshpande et al., 2019). This triggers the need for intelligent tool support that assists stakeholders in finding

dependencies by exploiting the textual descriptions of the requirements. To that end, a major research objective of this thesis is dedicated to address this important subject area.

(Q1.1) *How can we automatically identify dependency relations between textually-defined requirements?*

One major problem that most recommender systems have in common is the *cold-start problem* (Schein et al., 2002; Xu et al., 2015). In general, the cold-start problem describes situations where no or only very sparse information about an item, a user, or both, is available and thus no useful recommendation can be presented to the user. In the context of dependency detection, cold-start scenarios typically occur at the beginning of a project when no or only a few dependencies have been defined by the stakeholders. As mentioned before, the identification of requirement dependencies in large sets of textually-defined requirements is very challenging for stakeholders. The major reason for this is the number of possible requirement combinations / pairs that have to be analyzed – this number increases quadratically with the number of requirements. Due to the quadratic growth in the number of requirement pairs, it is challenging to get and collect complete dependency datasets from many different (large) software projects that consist of many requirements. However, situations with incomplete dependencies of the current project and few dependency datasets from past projects present a challenge for many existing dependency detection approaches. This underlines the importance to the development of proper solutions to tackle the cold-start issue of dependency detection tasks. This leads to the next research question.

(Q1.2) *What is a proper solution to handle cold-start issues in requirements dependency identification tasks?*

2. Recommendation support for group-based requirements prioritization

In addition to an extensive analysis of requirement dependency relationships, requirements prioritization represents another key prerequisite for successful release planning. In release planning, it is particularly important to make group decisions. Requirements prioritization represents the basis for a complete and sound release plan which is essential to successfully complete a software project. Requirements are often prioritized on the basis of one-dimensional utility estimates. In requirements engineering, there exist numerous techniques to prioritize requirements, for example, *binary search tree*, *planning game*, *numerical assignment technique*, or *analytic hierarchy process* (Sadiq et al., 2017; Qaddoura et al., 2017). However, these techniques have the effect that stakeholders see the requirements as rather simple abstract units during the prioritization process. This means that the priority of a requirement is only evaluated

on the basis of one dimension. Instead, it would be important that stakeholders are encouraged to take a closer look at the individual attributes of the requirements in order to obtain a multidimensional view on the requirements. This can result in a better estimation of the requirements' priority. Requirement attributes (also called *interest dimensions*) represent important metadata (information) to enrich the definition of a requirement (Firesmith, 2005). Examples of such attributes include the *time effort* to implement the requirement or the *risk* that the success of the project is jeopardized when the requirement is not considered in the project (Regnell et al., 2001). In this context, a relevant research topic that can be identified is to figure out whether and how the inclusion of additional requirement information in the evaluation (i.e., the evaluation across different dimensions of interest) can improve stakeholder evaluation behavior and prioritization quality. This raises the next research question.

(Q2.1) *How does the dimensionality of rating-schemes affect requirements evaluation behavior?*

In practice, requirements are often evaluated and prioritized by stakeholders during group meetings. A serious problem of such an approach is that, in many cases, only a small group of stakeholders actively contributes to the group conversation in a meeting (these are the opinion leaders). In such situations, the majority of the meeting participants hardly ever raise any objections against an argument provided by one of the opinion leaders. This can lead to two side effects that have to be mentioned here. On the one hand, the opinion leaders' arguments can influence the opinions of many other stakeholders, which then manifests itself in the form of undesired cognitive biases (such as the *anchoring effect*; see Stettinger et al., 2015) that negatively affect the group decision process. On the other hand, some decision-relevant information / knowledge may get lost since the (mostly inactive) rest of the stakeholder group hardly provides any valuable arguments due to their fear that they could appear uninformed or unsupportive, which is known to be part of the *groupthink* phenomenon (Janis, 1982). These two side effects result in reduced stakeholder interaction and less information exchange. However, a major precondition for high-quality group decisions is information exchange (Schulz-Hardt et al., 2006; Greitemeyer and Schulz-Hardt, 2003). As a consequence, mechanisms to improve stakeholder interaction and information exchange are essential to improve the quality of requirements prioritization. This paves the way for the next research question.

(Q2.2) *How can we increase stakeholder interaction in requirements evaluation to improve the quality of requirements prioritization?*

In most software projects, the number of requirements is high and the resources such as the time and costs to work on these requirements are limited. Therefore, not all requirements can be implemented. This requires the requirements managers to figure out which requirements are

of the highest (overall) relevance for the stakeholders. A common approach to achieve this is to prioritize the requirements in a group and to select the most appropriate candidates with a high priority on the final requirements. A detailed analysis and evaluation of these requirements represent the basis for a successful prioritization. In this context, evaluation approaches and user interfaces are needed that support the prioritization of requirements. In particular, it would be important to examine the influence that user interfaces have on the quality of the developed software product and which types of user interfaces are helpful to improve the quality of group-decisions in requirements prioritization. This gives rise to our next research question.

(Q2.3) In which way do different evaluation interfaces impact requirements prioritization and software quality?

3. Stakeholder Identification

After a successful prioritization of the requirements, a typical succeeding decision problem for requirements managers is to identify suitable stakeholders to whom the requirements can be assigned. As already mentioned, software systems are becoming more complex these days and often include large numbers of requirements. In a globalized world, large companies also have to pay more attention to the "truck factor" (Avelino et al., 2016), which describes the risk that important parts of a company's knowledge and expertise are only distributed among one or a few employees. Therefore, it is essential to ensure that decision-making power as well as stakeholder expertise are shared among several people in a company. However, in the context of traditional requirements engineering, a deep involvement and integration of the requirements managers in the requirements engineering process is common practice in many large companies. Apart from requirements prioritization, an important issue that has to be addressed in a software project is to assign responsibilities for requirements to be implemented. As stated by Hujainah et al. (2018), major limitations of available requirements prioritization approaches include scalability, complexity, and lack of automation and intelligence. While most existing recommendation approaches (Lim et al., 2010; Mobasher and Cleland-Huang, 2011; Alenezi et al., 2013) aim to address some of these limitations, we can still identify a need for more automated decision-support in requirements prioritization which reduces complexity and allows a higher level of scalability. Moreover, there also exists a need to support more complex stakeholder assignment processes in which not only individual actors but also large stakeholder groups and different departments are involved. This brings us to the next research question.

(Q3.1) How can we facilitate stakeholder assignment to support decision-makers?

Most existing recommendation solutions work well in traditional software development environments with clear stakeholder hierarchies (such as in public organizations or private sector companies), but they are less applicable in open-source projects where the software development process is more flexible and open to everyone. In open-source projects, the developers' interests can strongly vary regardless of whether the developers have just recently joined the project (newcomers) or have long-standing coding experience in the project (experts). A major task of open-source development is to constantly promote the commitment of the contributors and to take into account the individual interests as well as the different strengths and weaknesses of the developers. This is due to the reason that volunteers from all over the world can join and leave the communities at any point in time and there are no fixed working times. This introduces high fluctuations of contributors and the approach of how requirements are selected in open-source development follows a more first-come, first-serve principle. The high fluctuation of contributors is a common problem of most open-source projects – while a few contributors tend to dedicate their life to the project, many of them dig into the project and leave it after some time. Most developers find it difficult to find their way into these projects because they often fail at various initial hurdles during onboarding. A common issue at the beginning is to get an overview and the first (and correct) elements to start. But even for more experienced developers, the question arises on which requirements they should focus. Although existing works that address these open-source-specific issues suggest suitable requirements to developers, many of these solutions only focus on a specific target group (e.g., only newcomers or experienced developers) or on the whole developer community (Stanik et al., 2018; Tian et al., 2015). Other approaches offer solutions for a larger variety of contributors. Improved approaches, however, should try to close the aforementioned gaps by counteracting high developer fluctuations, as well as by providing help to more experienced developers. In this context, another topic for research can be derived in the terms of the following research question.

(Q3.2) How can we foster stakeholder engagement in open-source development while taking into account different stakeholder types (e.g., newcomers and experts)?

1.3. Contributions

The main contribution of this thesis is the provision of recommendation technologies that support and improve decision-making in requirements engineering. Based on the research questions defined in Section 1.2, we investigate various aspects and scenarios in requirements engineering where the application of recommendation technologies can lead to improvements in decision-making. The results of our investigations are based on the evaluation results of studies and experiments that we carried

out as part of the European research project OPENREQ². The contributions of this thesis are related to the research questions which were introduced in Section 1.2. Table 1.1 presents an overview of these contributions.

Research Questions	Contributions
(Q1.1) <i>How can we automatically identify dependency relations between textually-defined requirements?</i>	This contribution aims to promote the automated identification of requirement dependencies, since dependencies play an essential role in the creation of release plans. To answer this research question, we have evaluated a small real-world dataset that consists of 30 textually-defined requirements. In order to identify dependency relationships between requirements on a textual level, we have implemented two different content-based recommendation approaches. Both approaches were based on automated classification – the first approach was fed with TFIDF-features and the other used probabilistic features as input for training. To increase the comparability of the evaluation results, we have evaluated both approaches with a variety of different classifiers such as <i>Linear Support Vector Machine</i> , <i>k-Nearest Neighbors</i> , and <i>Random Forest</i> . The prediction quality of the different classifiers was compared, and an approach based on <i>Latent Semantic Analysis</i> was used as a baseline approach. The main outcome of our comparison (see Section 4.6) is that our recommendation approach based on <i>Random Forest using probabilistic features achieved the highest prediction quality of all evaluated approaches</i> .
(Q1.2) <i>What is a proper solution to handle cold-start issues in requirements dependency identification tasks?</i>	At the beginning of a project, the known set of existing requirement dependencies is small and limited. In such cases, classification-based approaches can be trained with dependency information extracted from other software projects that stem from the same domain. However, software projects are often filled with hundreds of requirements and these projects sometimes belong to a domain for

²OPENREQ: <https://openreq.eu>

	<p>which not enough dependency data exists to learn a classification-based recommendation model. For this reason, we have investigated how alternative text-based learning approaches are suitable to identify dependencies between requirements when no domain-specific data records are available. To that end, we have developed a recommendation approach based on <i>Latent Semantic Analysis</i> (LSA), which divides the text of the requirements on the basis of semantic characteristics and detects requirement dependencies based on the similarity of these characteristics. Our approach was evaluated with a real-world dataset and compared with several classification-based algorithms; related evaluation results are presented in Section 4.6.</p>
<p>(Q2.1) <i>How does the dimensionality of rating-schemes affect requirements evaluation behavior?</i></p>	<p>During requirements prioritization, requirements are often viewed as single one-dimensional abstract units. However, requirements usually contain additional meta-information (attributes) that can be used as evaluation criteria to extend a one- to a multidimensional rating scheme. To compare the evaluation behavior of stakeholders between one- and multidimensional rating schemes, we have conducted a large-scale user study with 313 participants. The participants worked in groups and the OPENREQ!LIVE platform was used to support the groups with the task of defining and prioritizing requirements for the development of a software project. Following a between-subjects study design approach (Charness et al., 2012), the groups had to use either a one- or a multidimensional rating scheme to evaluate and prioritize the requirements. Our study results indicate that stakeholders tend to evaluate requirements more frequently and critically when they are confronted with multidimensional (instead of one-dimensional) rating schemes (see Section 5.4).</p>
<p>(Q2.2) <i>How can we increase stakeholder interaction in requirements evaluation to improve the quality of requirements prioritization?</i></p>	<p>In requirements engineering, requirements are often evaluated by groups of stakeholders. The aggregation of different stakeholder evaluations constitutes a group decision which benefits from the variety of different opinions (Schulz-Hardt et al., 2006). Moreover, mechanisms that</p>

lead to more contrary opinions can trigger more information exchange among stakeholders and tweak the outcome of requirements prioritizations (Schulz-Hardt et al., 2006; Al-Rawas and Easterbrook, 1996; Coughlan and Macredie, 2002). To that end, we introduced an argumentation-based evaluation approach that allows stakeholders to create arguments for and against requirements (instead of evaluating them numerically). The main idea of argumentation-based evaluations is to encourage stakeholders to deal more intensively with the requirements in order to trigger more discussions. To measure the effect of information exchange, we conducted a user study with 313 students who worked in small groups (4-6 group members) on the OPENREQ!LIVE platform to define and prioritize requirements for the development of a software product. The groups had to evaluate the requirements using either numeric ratings or arguments. One key finding of the evaluation is that groups which evaluated requirements using arguments interacted with the system and adapted the requirements more often than other groups using numeric evaluation interfaces - further results are presented in Section 5.4.

(Q2.3) *In which way do different evaluation interfaces impact requirements prioritization and software quality?*

To investigate the impact of rating interfaces on the quality of the software product, we compared three user interfaces with different (one- and multidimensional) rating schemes. In a user study, 313 computer science students had to work in groups to define and prioritize the requirements of a software project using OPENREQ!LIVE, and to develop the software product based on the prioritized list of finally selected requirements. The groups were randomly assigned to one of three user interfaces and used the user interface to evaluate the requirements. The set of rating schemes included a basic one-dimensional, a basic multidimensional, and an argumentation-based multidimensional rating scheme. To assess the quality of the developed software products, we analyzed the grades the students have received for the final software product as well as the number

	<p>of completed requirements. A comparison of the groups (see Section 5.4) reveals that a higher fraction of completed requirements as well as a higher quality of the final software product can be observed for the groups which used the multidimensional rating interfaces.</p>
<p>(Q3.1) <i>How can we facilitate stakeholder assignment to support decision-makers?</i></p>	<p>In this contribution, we investigate how to accelerate and improve stakeholder assignment tasks in traditional requirements engineering scenarios. In Chapter 6 of this thesis, we propose a recommendation environment which follows the approach of distributing the stakeholder assignment task among several stakeholders. The recommendation environment includes two recommender systems which provide guidance on the stakeholder selection procedure throughout the entire stakeholder assignment process. Our approach aims to minimize the workload of requirements managers and is particularly useful for large and complex software projects. This work was regarded as one of the best contributions at the <i>International Workshop on Configuration Systems (ConfWS 2018)</i> and for this work we received the <i>Siemens Best Student Paper Award</i>.</p>
<p>(Q3.2) <i>How can we foster stakeholder engagement in open-source development while taking into account different stakeholder types (e.g., newcomers and experts)?</i></p>	<p>In order to promote stakeholder engagement in open-source software development, we developed a content-based recommender system. The recommender system provides support in the time-consuming search for suitable requirements by proposing requirements to contributors (stakeholders) which are tailored to their personal preferences and skills. We have published the recommendation environment together with a plugin for the ECLIPSE community which presents the recommended requirements to stakeholders and allows them to interact with the recommendations. The underlying recommendation approach creates user profiles for developers from resolved requirements and proposes new requirements with the support of classification algorithms. To take into account the individual characteristics and working methods of the different stakeholder types, we optimized the recommendation</p>

model to focus on precision rather than on recall. This basically means that the model makes fewer but more correct recommendations (quality over quantity) instead of many recommendations where more predictions are incorrect (quantity over quality). We have evaluated our approach with three large datasets from open-source projects including ECLIPSE, MOZILLA, and LIBREOFFICE. Section 8.4 presents the evaluation results. The results are promising and indicate that our prediction models are able to recommend suitable requirements for stakeholders with a high level of prediction quality. For this work, we received the *Runner Up for the Best Student Paper Award* at the International Conference on Web Intelligence (WI 2019).

Table 1.1.: Contributions of this thesis with regard to the corresponding research questions.

1.4. Thesis Outline

This thesis consists of ten chapters. The documentation of related and future work is organized according to the subject matter and divided into topic-specific sections. These sections are assigned to the respective chapters based on their topic and are included in these chapters. The thesis is structured as follows.

Chapter 1 gives an introduction and a motivation for the main topics of this thesis. This chapter also summarizes and consolidates the relevant research questions, and highlights the main research contributions of this thesis. The chapter is concluded with a brief overview of the structure of this thesis.

Chapter 2 introduces common choice scenarios that occur in the context of group decisions. In this chapter, we shed light on a detailed categorization of choice scenarios along the dimensions of knowledge representation and the inclusion of constraints. In addition, we also provide some examples that demonstrate how to determine group recommendations. The content of this chapter represents essential background knowledge for the following chapters of this thesis. For example, the choice scenario *configuration* affects the recommendation solutions discussed in Chapter 4. The recommendation approaches presented in Chapters 5, 7, and 8 cover a combination of different choice scenarios including *ranking*, *release planning*, and *configuration*. Moreover, the recommendation environment introduced in Chapter 6 represents a combination of the choice scenarios *resource balancing* and *configuration* implemented as a multi-stage decision process.

Chapter 3 provides an overview of research in the field of recommender systems in requirements engineering. In this chapter, we introduce suitable recommendation approaches for requirements engineering. Furthermore, we give an overview of existing recommendation solutions that address common requirements-engineering-related tasks and present application scenarios where these solutions can be applied. In addition, the chapter also presents selection criteria that can help to find suitable solutions to be applied in different application environments. The chapter ends with a conclusion and a discussion of open research issues.

After an overview of recommender systems in the field of requirements engineering has been conveyed, the subsequent chapters present our research topics and research questions that were examined in this thesis. An in-depth analysis of various research areas in the field of requirements engineering has shown that the detection of requirement dependencies represents an open research area where there is still plenty of space left for improved decision-support. Moreover, this area plays a crucial role for subsequent requirements engineering tasks and thus represents our first focus of the thesis. In **Chapter 4**, we propose two novel recommendation approaches to identify dependency relationships between requirements. Within the scope of a user study, we evaluated a real-world dataset by combining expert knowledge from requirements engineering practitioners with crowd-knowledge from study participants in order to obtain a reliable ground truth of requirement dependency data that was then used to train and evaluate our approaches.

An extensive analysis and identification of all requirement dependencies represents the prerequisite for a successful prioritization of requirements. In **Chapter 5**, we focus on this topic and compare three group recommender user interfaces to support the prioritization of requirements. Our recommendation approach aims to increase stakeholder engagement by involving the whole stakeholder group in the decision-making process. This is achieved by encouraging the stakeholders to evaluate the requirements using the presented user interfaces. As part of a large-scale user study, the three user interfaces were compared. In this user study, the participants had to work in small groups and develop a software product. The groups could either use a one-dimensional 5-star rating interface, a multi-dimensional rating interface, or a multidimensional argument-based rating interface to evaluate and prioritize requirements. We analyzed the rating behavior of the users more precisely and measured the general impact of the prioritization results on the software quality.

In the context of requirements prioritization, the assignment of requirements to suitable stakeholders represents another related follow-up decision problem for requirements managers. **Chapter 6** introduces a group-based recommendation solution to support requirements managers in this complex decision problem. The discussed approach facilitates the decision process of stakeholder assignment by fostering group-based decision-making. The recommendation environment includes two recom-

mentation services which provide guided decision support throughout the whole decision process. Further support in the form of stakeholder evaluations / ratings helps to involve the whole stakeholder group in the decision-making process and is important in order to achieve time savings and a reduced involvement of requirements managers.

In addition to conventional industrial software development, the field of open-source development defines another interesting research area that needs to be addressed. In **Chapter 7**, we propose a utility-based recommendation approach for open-source communities to provide decision-support in stakeholder assignment tasks. The primary role of our approach is to support open-source developers in the identification of the most relevant and interesting requirements³ to be implemented next. To demonstrate the potential of our approach, we use the ECLIPSE community as a showcase example. The approach generates recommendations that are tailored to the interests of the developers and the ECLIPSE community in order to avoid time-consuming and inefficient search processes. Moreover, we also show how to model and integrate requirement dependencies into such utility-based requirements prioritization processes.

In contrast to Chapter 7 where the main focus lies on the perspective of the developer community, **Chapter 8** presents a content-based approach that suggests requirements³ which are tailored to the individual preferences of the developers. In this chapter, we also introduce a plugin for ECLIPSE developers that presents the recommended list of requirements and allows them to give feedback on the recommendations. Our approach uses supervised classification techniques to predict and recommend suitable requirements to a developer. We have evaluated our approach with three classifiers (*Multinomial Naïve Bayes*, *Decision Tree*, *Random Forest*) and compared the prediction quality of all classifiers.

The recommendation approaches presented in the previous chapters were developed and evaluated as part of the European research project OPENREQ. **Chapter 9** gives an overview of intelligent recommendation and decision tools which were developed within the scope of the OPENREQ project. In this context, we also present OPENREQ!LIVE which is a user-friendly requirements engineering platform that provides users central access to the developed recommendation tools. Beyond that, OPENREQ!LIVE fosters the cross-fertilization of ideas between stakeholders by empowering them to take advantage of the full recommendation power, and to work together on requirements in a collaborative way. Finally, the chapter summarizes the study results and the major outcomes of the project.

Chapter 10 concludes this thesis and presents an overview of open research issues.

³In the specific context of open-source development, requirements are usually considered as "issues" or "bugs".

Choice Scenarios Beyond Basic Recommendation

The contents of this chapter are based on Felfernig et al. (2018). The author of this thesis provided the design of the working samples that are presented throughout the chapter.

2.1. Abstract

In this chapter, we present different choice scenarios that typically occur outside the scope of basic recommendations. In addition to choice scenarios in which a group recommender selects items from a set of explicitly defined (enumerated) items (e.g., the selection of a restaurant for a dinner, or the selection of a holiday destination), further choice scenarios exist. These choice scenarios differ in the way alternatives are represented and recommendations are determined. In this context, we introduce a categorization of these scenarios and discuss knowledge representation as well as group recommendation aspects on the basis of examples.

2.2. Introduction

In this chapter, we analyze scenarios that go beyond the ranking and selection of explicitly defined items (alternatives). We first characterize these scenarios with regard to the aspects of (1) the *inclusion of constraints* (constraints allow the definition of restrictions regarding the combination of choice alternatives) and (2) the *approach to define alternatives* (alternatives can be either represented *explicitly* or in terms of *parameters*). Thereafter, we discuss these scenarios in more detail on the basis of examples. There are hierarchical relationships between some scenarios. Examples thereof are *release planning*, *triage*, *resource balancing*, and *sequencing*. These scenarios can be considered as subtypes of *configuration* differing in the type of variables and constraints used. We also differentiate between (1) *basic choice problems* (*ranking*, *packaging*, *parametrization*, *configuration*,

release planning, resource balancing, sequencing, and triage) and (2) methods for getting people’s input concerning choice problems (voting, questionnaires, and parametrization). The choice scenarios introduced in this chapter are the following (see Figure 2.1).

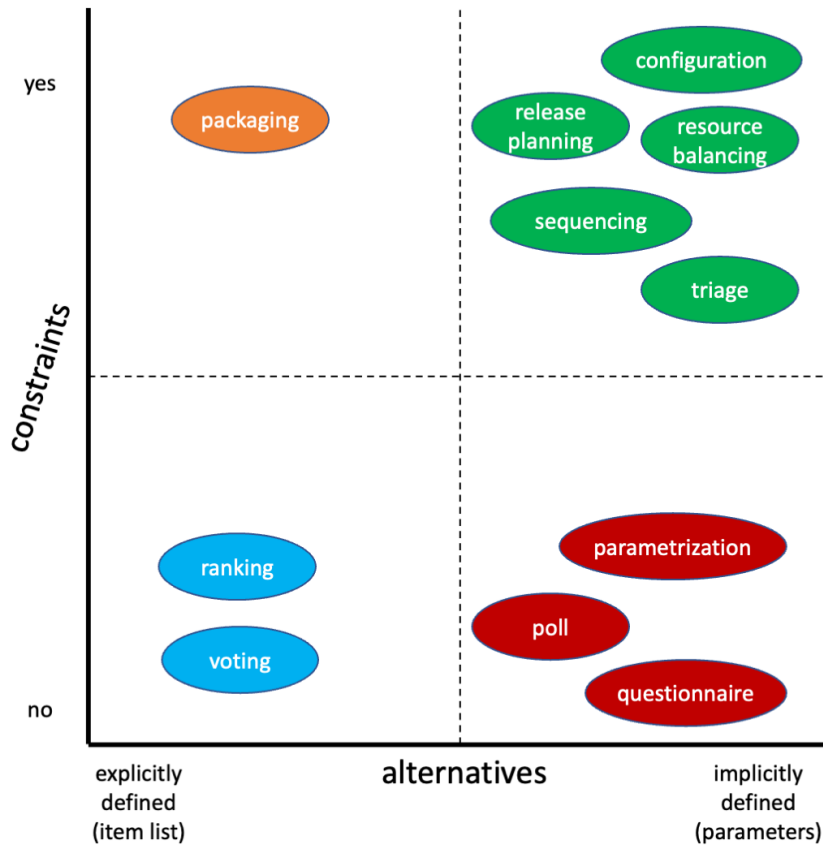


Figure 2.1.: Choice scenarios categorized with regard to (1) *constraint inclusion* and (2) the *representation of alternatives* (as parameters or items).

Ranking. The overall goal of *ranking* is to derive a ranked list of items as a recommendation for a group. Ranking scenarios typically do not include constraints and choice alternatives are represented in the form of a list of explicitly defined items, for example, *restaurants* or *holiday destinations*.

Packaging. Package recommendation goes beyond basic ranking (Qi et al., 2016, 2017; Xie et al., 2010). The overall goal is to recommend combinations of items while taking into account constraints that restrict the way in which different items can be combined. For example, in *holiday trip planning*, a package recommendation problem is to find a set of destinations for the group that takes into account global constraints such as upper price limit and maximum total distance between the destinations, but also constraints related to individual items. For example, specific destinations should be excluded,

or either one or the other should be visited but not both. Items in packaging problems are specified explicitly, for example, a list of museums and a list of restaurants. Another example of packaging is a group decision regarding the composition of a *Christmas party menu*. Decision alternatives are represented by lists of menu items where each item is associated with one of the categories *starter*, *main dish*, and *dessert*. Constraints can be specified, for example, according to the maximum number of menu items and the upper price limit of a menu.

Parametrization. Parametrization decisions are related to detailed aspects of an item – related alternatives are represented as parameter values. In parametrization, no restrictions exist between the parameter values. In the context of group decision making, an example is the *parametrization of an already selected travel destination* or the *parametrization of intended properties of an already selected hotel*. Examples of parameters of a travel destination are number of days to be spent at the destination and time of the year. Parameters describing intended properties of hotels are the availability of a beauty farm, whirlpool, fitness studio, and massage service (Jameson et al., 2004).

Configuration. Configuration (Aldanondo and Vareilles, 2008; Felfernig et al., 2014; Stumptner, 1997) is one of the most successful applications of artificial intelligence techniques. In terms of knowledge representation, configuration scenarios are similar to parametrization, i.e., decision alternatives are represented in terms of parameters. In contrast to parametrization, configuration tasks include a set of constraints that restrict the combination of individual parameter values. Examples thereof are the group-based configuration of *smart home installations* and the group-based configuration of a *car* (e.g., a new company car) (Felfernig et al., 2016; Leitner et al., 2016). Further examples of group-based configuration are *release planning* (Felfernig et al., 2011), *resource balancing*, *sequencing*, and *triage*. Because of their wide-spread application, these scenarios will be discussed in separate subsections.

Release Planning. Both, in terms of knowledge representation and inclusion of constraints, a release planning task is a specific type of configuration task (Ninaus et al., 2014). In software engineering, release planning refers to the task of assigning a set of requirements to one of a defined set of releases. This scenario is usually a group decision scenario, since stakeholder groups engaged in a software project have to make release-related decisions. An example of a related constraint is: *since the overall effort is too high, requirement x and requirement y must not be implemented in the same release*.

Triage. Similar to release planning, triage can be considered a specific type of configuration task. Triage decisions can occur in domains such as medical decision making and software engineering. The overall goal of the underlying decision is to determine a tripartition¹ of a given set of alternatives. In *early requirements engineering* (Ninaus et al., 2014), triage can be applied to figure out (1) require-

¹We limit our discussions to scenarios with three partitions.

ments that are essential for a company and must be implemented immediately, (2) requirements that can be implemented if the resources are available, and (3) unimportant requirements with no need for implementation in the near future. As opposed to this, the focus of release planning is to decide a.o. about the time of implementation. Constraints are similar to those occurring in the context of release planning. Further examples of triage-based decisions are *selection and assignment of students to open research projects of a research group* (students with high potential should be preferred, students with a low probability of successfully completing their tasks should be assigned to standard projects but not research projects, and all other students should receive a research project position if possible), *funding decisions* (distribute the available budget between high-potential projects while taking into account an upper funding limit, do not fund low-potential projects, and fund 'in-between' projects if additional money is available), *idea management* (focus on high-potential ideas, filter out low-potential ideas, and take into account ideas 'in-between' if the needed resources are available), and *product line scoping* (Schmid, 2000) (include the most relevant product features, features with potentials for new markets if possible, and filter out low-potential ones).

Resource Balancing. The goal of resource balancing is to assign *consumers* to *resources* in such a way that a given set of constraints is satisfied. In this context, consumers and resources can represent humans as well as physical equipment or software. The assignment of resources to consumers can be represented in terms of parameters. Resource balancing often includes a set of constraints, for example, each student should be assigned exactly one paper and paper assignments should be equally distributed. Thus, resource balancing can also be interpreted as a specific kind of configuration task. In configuration scenarios, resource balancing is often included as a subtask, for example, to balance power supply and consumption (Felfernig et al., 2014).

Sequencing. Sometimes, alternatives have to be arranged in a sequence. For example, when *planning a trip around the island of Iceland*, the sequence of venues (when to visit which destination) has to be clear from the outset since hotel reservations have to be arranged correspondingly. Items in sequencing tasks are often represented in terms of parameters. Constraints are related to user preferences (e.g., three waterfalls should not be visited directly one after another) and further restrictions (e.g., the distance between two destinations in a sequence should be below 100 kilometers and the overall length of the round trip should be minimized).

Polls and Questionnaires. Polls and questionnaires are basic means to better understand the opinions of a group or a community. Thus, both can be considered as basic decision support mechanisms. In poll scenarios, the group giving the feedback is in many cases not directly engaged in a decision making process. Polls are defined in terms of a question (parameter) and possible answers. No constraints are defined with regard to the choice alternatives. Questionnaires are a concept similar to polls with the difference that more than one question is typically posed and new questions are sometimes

selected depending on answers that have already been provided.

Voting. Compared to questionnaires and polls, voting has a strong decision aspect, since a group or a community decides on which alternative(s) should be chosen (Levin and Nalebuff, 1995). This takes place on the basis of a predefined process. The underlying options are represented in an explicit fashion, like presidency candidates or candidate soccer players for the 'goal of the month'. In voting, there are no constraints regarding the alternatives.²

Due to the high diversity of existing choice scenarios, we do not claim completeness. The scenarios presented must be seen as examples, i.e., different variants thereof exist. In the following, we will discuss knowledge representations of the choice scenarios shown in Figure 2.1, and sketch approaches to include group recommendation techniques.

2.3. Ranking

In basic ranking scenarios (Felfernig et al., 2014), *choice alternatives are enumerated and no constraints* are applied to the alternatives. A group's task is to identify a ranking and then select *one item* (e.g., in the context of selecting a restaurant for dinner or a logo for a new product) or *a couple of items* (e.g., when selecting the n best conference papers or selecting the n best proposals submitted to a funding organization). Alternatives do not necessarily need to be specified completely before the decision process starts, for example, in *idea competitions* and *open innovation* scenarios, alternatives can be added during the decision process. A simple example of a ranking scenario is depicted in Table 2.1. Each item t_i received one ranking per group member. A score is associated with each rank, for example, rank 1 receives 3 points, rank 2 receives 2 points, etc. The item with the highest *borda count* (BRC) scoring is recommended (in our case item t_4 which is indicated with \surd in Table 2.1).³

Item	ranking (score)			BRC	ranking
	u_1	u_2	u_3		
t_1	4 (0)	4 (0)	4 (0)	0	4
t_2	2 (2)	3 (1)	2 (2)	5	2
t_3	3 (1)	2 (2)	3 (1)	4	3
t_4	1 (3)	1 (3)	1 (3)	9	1 \surd

Table 2.1.: A basic group-based ranking scenario. Group members u_i provide ranks for items $t_i \in I$ (alternatively, rankings can be derived by a recommender). Thereafter, an aggregation function such as *borda count* (BRC) can be used to derive a corresponding ranking for the group. The \surd symbol indicates the recommended item.

²For a discussion of the potential impacts of voting strategies we refer to Levin and Nalebuff (1995).

³The aggregation functions used in this and other scenarios are considered as convenient, however, other alternatives might exist.

2.4. Packaging

In a packaging scenario (see Table 2.2) (Qi et al., 2016, 2017), each item t_{ij} is associated with a specific item type i . Choice alternatives are explicitly defined per item type and constraints related to the alternatives have to be taken into account. A group has to select items of different item types and compose these into a corresponding package. An example of a constraint that is defined in such a scenario is: *the number of selected items per item type must be exactly 1* (see constraint c_1 in Table 2.2). Table 2.2 depicts an example of a group-based packaging scenario. Each item receives a ranking per group member and the item with the highest *borda count* (BRC) score within a specific item type i is the group recommendation for item type i . The recommended package in our example is $\{t_{11}, t_{21}, t_{31}\}$. In some scenarios, more than one item per item type is requested or less items than defined types are allowed to be included in a package recommendation. In more complex scenarios, constraints are also specified at the individual item level. An example of such a constraint is an incompatibility between the items t_{22} and t_{33} , i.e., these items must not be part of the same package. In the case of such constraints, solution search in packaging scenarios can be implemented on the basis of conjunctive (database) queries.

	item ranking (score)								
	item type 1			item type 2			item type 3		
	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{23}	t_{31}	t_{32}	t_{33}
u_1	1 (3)	2 (2)	3 (1)	2 (2)	1 (3)	3 (1)	1 (3)	2 (2)	3 (1)
u_2	2 (2)	3 (1)	1 (3)	1 (3)	2 (2)	3 (1)	1 (3)	2 (2)	3 (1)
u_3	1 (3)	2 (2)	3 (1)	1 (3)	2 (2)	3 (1)	3 (1)	2 (2)	1 (3)
<i>BRC</i>	8	5	5	8	7	3	7	6	5
type-wise ranking	1√	2	2	1√	2	3	1√	2	3
$c_1 : \forall i : \#proposeditems(type\ i) = 1$									

Table 2.2.: A group-based packaging scenario. Users provide ranks for items t_{ij} (j^{th} item of type i). Thereafter, an aggregation function such as *borda count* (BRC) can be used for deriving a proposed package (in our case, $\{t_{11}, t_{21}, t_{31}\}$). The \sqrt symbol indicates the recommended items part of the package.

2.5. Parametrization

The alternatives are defined in terms of *parameters* and there are *no constraints* related to the alternatives. In such a scenario, a group's task is to select one value per parameter. An example of a group-based parametrization scenario is presented in Table 2.3. Each group member specifies his / her preferences with regard to the different parameters and then the values that were selected in the majority of the cases are considered as candidates for the group recommendation. The recommendation (parametrization) in our example is $\{par_1 = a, par_2 = 1, par_3 = 2\}$.

parameter	preferences			MAJ
	u_1	u_2	u_3	
$par_1(a, b, c)$	a	a	c	a \checkmark
$par_2(1, 2, 3)$	1	1	1	1 \checkmark
$par_3(1, 2)$	2	2	1	2 \checkmark

Table 2.3.: Group-based parametrization. Users define preferences with regard to the parameters par_i . Thereafter, an aggregation function such as *majority voting* (MAJ) can be used for recommending a parametrization (in our case, $\{par_1 = a, par_2 = 1, par_3 = 2\}$). The \checkmark symbol indicates recommended parameter values.

2.6. Configuration

In group-based configuration scenarios (Felfernig et al., 2016), the alternatives are defined by parameters and corresponding domain definitions. In most configuration scenarios, constraints restrict possible combinations of parameter values. Similar to parametrization scenarios, a group's task is to select one value per parameter such that the set of parameter value assignments is consistent with the defined constraints (Felfernig et al., 2014). An abstract example of a group-based configuration scenario is shown in Table 2.4. Each group member specifies his / her preferences with regard to the values of the parameters $\{par_1, \dots, par_4\}$. An example constraint is $c_1 : par_3 = u \rightarrow par_4 = 1$. Table 2.4 also depicts the solution candidates, i.e., complete sets of parameter assignments that take into account the defined constraints. These configurations include *trade-offs* in terms of neglecting some of the user preferences due to the fact that the union of all user preferences would be inconsistent (Felfernig et al., 2012). In our example shown in Table 2.4, *least misery* (LMS) is applied to evaluate the configuration candidates (to determine a recommendation). *Misery* in this context is defined as the *number of times the preferences of an individual user are not taken into account* by a configuration. In contrast to rating-based approaches, the higher the value the lower the quality of the corresponding configuration.

parameter	preferences			configuration (solution)					misery			LMS
	u_1	u_2	u_3	id	par_1	par_2	par_3	par_4	u_1	u_2	u_3	
$par_1(a, b, c)$	a	a	c	1	a	1	u	1	1	1	1	1 \checkmark
$par_2(1, 2)$	1	1	1	2	c	1	u	1	2	2	0	2
$par_3(u, v)$	u	u	u	3	b	1	u	1	2	2	1	2
$par_4(1, 2)$	2	2	1	$c_1 : par_3 = u \rightarrow par_4 = 1, c_2 : par_2 \neq 2, c_3 : par_3 \neq v$								

Table 2.4.: A group-based configuration scenario. Users u_i specify their preferences in terms of parameter values. Constraints c_i specify the restrictions, a configuration must take into account. Thereafter, an aggregation function such as *least misery* (LMS) can be used for deriving a recommended configuration (in our case, $\{par_1 = a, par_2 = 1, par_3 = u, par_4 = 1\}$). The \checkmark symbol indicates the configuration parameter values recommended to the group.

Solving Configuration Tasks. Configuration tasks can be solved using constraint solvers (Felfernig et al., 2014; Tsang, 1993). Thus, constraint solvers take over the role of determining candidate recommendations. These solvers generate solutions (candidate recommendations) consistent with the defined set of constraints. Due to the combinatorial explosion, it is often not possible to generate all possible solutions and then to filter out the best ones by using an aggregation function (Falkner et al., 2011). In order to deal with such situations, *search heuristics* that help to increase the probability of finding solutions that are optimal with regard to a selected aggregation function must be integrated into the constraint solver. A more lightweight integration of aggregation functions can be achieved with *majority voting* (MAJ). The votes of group members can be applied to derive preferences (Alanazi et al., 2012). For example, for par_1 we can derive a preference ordering $a \succ c \succ b$ indicating that a is preferred by a majority of group members (over c and b) and that c is preferred over b . Such preferences can be directly encoded as variable (domain) orderings into a constraint solver (Polat-Erdeniz et al., 2017).⁴

2.7. Release Planning

Release planning is a configuration task (Ninaus et al., 2014) where the *alternatives* (possible assignments of requirements to releases) *are defined as parameters* and corresponding domain definitions. In most release planning scenarios, *constraints* restrict the possible assignments of requirements to releases. A group’s task is to find one value per parameter (each requirement needs to be assigned to a release) in such a way that all assignments are consistent with the defined constraints. An example of a group-based release planning task is shown in Table 2.5.

parameter	preferences			release plan					misery			LMS
	u_1	u_2	u_3	id	req_1	req_2	req_3	req_4	u_1	u_2	u_3	
$req_1(1..2)$	1	1	2	1	1	1	2	2	1	0	4	4
$req_2(1..2)$	1	1	2	2	1	2	1	2	1	2	2	2 \checkmark
$req_3(1..2)$	1	2	1	3	2	1	1	2	1	2	2	2 \checkmark
$req_4(1..2)$	2	2	1	4	2	2	1	1	3	4	0	4

$c_1 : req_3 \leq req_4, c_2 : \forall_i : numreqrel_i \leq 2$

Table 2.5.: A group-based release planning scenario. Users can specify their preferences in terms of assignments of requirements (req_i) to releases. Additionally, constraints c_i specify properties a release plan must take into account. Thereafter, an aggregation function such as *least misery* (LMS) can be used for deriving a proposed release plan (in our case, for example, release plan 2). The \checkmark symbol indicates recommended release plans.

Each group member specifies his / her preferences with regard to the assignment of requirements to releases. Example constraints are $c_1 : req_3 \leq req_4, c_2 : \forall_i : numreqrel_i \leq 2$ which denote the fact

⁴For example, <https://choco-solver.org>.

that (1) requirement req_3 must not be implemented after requirement req_4 and (2) no more than two requirements should be assigned to the same release. Similar to the aforementioned configuration scenario, the preferences of individual users are aggregated using *least misery* (LMS). In this context, LMS denotes the maximum number of times the preferences of an individual user are neglected by a release plan. For example, release plan 1 ignores the preferences of user u_3 *four times* which is the maximum for release plan 1. Both release plan 2 and 3 have the lowest LMS. Consequently, release plans 2 and 3 can be recommended. Techniques that can be used to determine individual release plans are the same as those discussed in the context of solving configuration tasks.

2.8. Triage

Triage can be regarded as a configuration task. In the context of *software requirements engineering*, alternative requirements have to be assigned to one of the three triage categories: *accept* (a) = requirement must be implemented, *maybe accept* (m) = requirement can be implemented if resources are available, and *reject* (r) requirement will not be implemented (now). As in release planning, constraints can restrict the assignment of requirements to the three categories. Table 2.6 includes an example of a simple triage task.

parameter	preferences			triage					misery			LMS
	u_1	u_2	u_3	id	req_1	req_2	req_3	req_4	u_1	u_2	u_3	
$req_1(a, m, r)$	a	r	a	1	a	m	a	m	2	2	2	2 \checkmark
$req_2(a, m, r)$	r	m	r	2	a	r	a	r	0	4	0	4
$req_3(a, m, r)$	a	r	a	3	m	a	m	a	4	4	4	4
$req_4(a, m, r)$	r	m	r	4	r	a	r	a	4	2	4	4

$$c_1 : req_1 = req_3, c_2 : req_2 = req_4$$

$$c_3 : a(req_1) + a(req_2) + a(req_3) + a(req_4) = 2$$

Table 2.6.: Group-based triage. Users specify their preferences by categorizing requirements (req_i) into *a* (accept), *m* (maybe accept), and *r* (reject). Constraints c_1 and c_2 specify dependencies between requirements, c_3 specifies that two requirements have to be accepted (*a*). An aggregation function such as *least misery* (LMS) can be used for deriving a triage solution (in our case, triage 1). The \checkmark symbol indicates the triage recommended to the group.

A group's task is to assign one category to each requirement in such a way that all assignments are consistent with the defined constraints. In this example, the proposed triage follows the recommendation determined by *least misery* (LMS). Techniques that can be used to determine individual triage solutions are the same as those discussed in the context of solving configuration tasks.

2.9. Resource Balancing

A resource balancing task is defined on the basis of *parameters* $r_i u_j$ indicating the assignment of a consumer (user) u_j to a resource r_i ($r_i u_j = 1 \leftrightarrow$ consumer (user) j is assigned to resource i). In the example given in Table 2.7, each consumer (user) u_j provided a preference evaluation (on a scale 1..5) with regard to all potential assignments $r_i u_j$.⁵ The outcome is a *resource assignment* that indicates which consumer is assigned to which resource(s). In our example, resource balancing is interpreted in such a way that *each resource should be assigned to nearly the same number of consumers* and *each consumer should be assigned to exactly one resource* (see constraints c_1 – c_4 in Table 2.7; nr_i are parameters / variables representing the quantity of users assigned to resource i).

parameter	preference rating ($r_i u_j$)	resource assignment (rating)						LMS	
		id	$r_1 u_1$	$r_1 u_2$	$r_1 u_3$	$r_2 u_1$	$r_2 u_2$		$r_2 u_3$
$r_1 u_1(0, 1)$	5	1	1 (5)	1 (5)	0	0	0	1 (2)	2
$r_1 u_2(0, 1)$	5	2	1 (5)	0	1 (4)	0	1 (1)	0	1
$r_1 u_3(0, 1)$	4	3	1 (5)	0	0	0	1 (1)	1 (2)	1
$r_2 u_1(0, 1)$	4	4	0	1 (5)	1 (4)	1 (4)	0	0	4 \checkmark
$r_2 u_2(0, 1)$	1	5	0	1 (5)	0	1 (4)	0	1 (2)	2
$r_2 u_3(0, 1)$	2	6	0	0	1 (4)	1 (4)	1 (1)	0	1

$$c_1 : nr_1 = r_1 u_1 + r_1 u_2 + r_1 u_3$$

$$c_2 : nr_2 = r_2 u_1 + r_2 u_2 + r_2 u_3$$

$$c_3 : |nr_1 - nr_2| \leq 1$$

$$c_4 : r_1 u_1 + r_2 u_1 = 1 \wedge r_1 u_2 + r_2 u_2 = 1 \wedge r_1 u_3 + r_2 u_3 = 1$$

Table 2.7.: Group-based resource balancing. Users specify their preferences with regard to resource assignments in terms of *ratings*. Constraints c_i specify properties a resource assignment must take into account. *Least misery* (LMS) denotes the lowest user-specific evaluation of a resource assignment. The \checkmark symbol indicates the recommended assignment (in our case, assignment 4).

Choice scenarios similar to resource balancing in terms of the used knowledge representation are *task assignment* (e.g., a set of tasks has to be assigned to the members of a group) and *production scheduling* (e.g., a set of orders has to be assigned to machines taking into account the preferences of different customers).

2.10. Sequencing

Sequencing can be regarded as a configuration task where sequential numbers have to be assigned to items. As in configuration, constraints can restrict the assignment. Table 2.8 depicts an example of a sequencing task. A group's task is to assign one sequential number to each item in such a way that all assignments are consistent with the defined constraints (in our case c_1). If sequences have

⁵In order to reduce evaluation efforts, a user could specify only preferred items and the system would assume negative evaluations for items a user did not evaluate.

already been pre-defined, sequencing can also be implemented as a ranking task where users evaluate sequences and an aggregation function determines the recommendations. An example thereof is shown in Table 2.9.

parameter	preferences			sequence				misery			LMS
	u_1	u_2	u_3	id	t_1	t_2	t_3	u_1	u_2	u_3	
$t_1(1..3)$	1	1	1	1	1	2	3	2	0	2	2 ✓
$t_2(1..3)$	3	2	3	2	1	3	2	0	2	0	2 ✓
$t_3(1..3)$	2	3	2	3	2	1	3	3	2	3	3
				4	2	3	1	2	3	2	3
				5	3	1	2	2	3	2	3
				6	3	2	1	3	2	3	3

$$c_1 : \forall u_i : u_i t_1 = x \rightarrow u_i t_2 \neq x \wedge u_i t_3 \neq x \dots$$

Table 2.8.: Group-based sequencing. Users specify their preferences in terms of assignments of sequential numbers to items t_i . Additionally, constraints c_i specify properties a sequence must take into account. Here, $u_i t_j$ is a parameter representing a user's (u_i) assignment of item t_j to a specific sequence position. *Least misery* (LMS) denotes the number of times, a user preference is neglected by a sequence. Sequences $id = 1$ and $id = 2$ can be regarded as recommendation candidates.

Different aspects of sequencing have been investigated by Masthoff (2004) in the context of selecting television items (e.g., news and commercials). In the scenarios investigated until now, the primary inputs for determining recommendations are the ratings provided by individual group members. However, as mentioned in Masthoff (2004), a group member's evaluation of an item does not only depend on his / her personal preferences, but also on the context in which the item is shown. The evaluation of an item also depends a.o. on a user's mood. For example, in the context of TV commercials, it is often the case that viewers prefer to see sad commercials in the middle of sad TV programs and humorous commercials are preferred in humorous programs. This indicates a need for *consistency*, i.e., users try to maintain a specific mood throughout a TV program (Masthoff, 2004). Masthoff presents an in-depth analysis of different influence factors in group decision making in the context of sequencing. Particularly, different social choice functions are compared with regard to their applicability in the domain of television item sequencing. Results of the presented studies show that group members try to avoid individual misery and care about fairness in group decision making. Interestingly, ratings are used in a non-linear way, i.e., differences between extreme values are considered higher compared to rating values near the average. For further related details we refer to Masthoff (2004). Due to the possibility of compensating for items that are perceived suboptimal with better ones, especially in the context of sequencing, it is usually possible to make sure that no one is miserable.

sequence				evaluation			AVG
id	t_1	t_2	t_3	u_1	u_2	u_3	
1	1	2	3	5	4	3	4 ✓
2	1	3	2	3	3	5	3.67
3	2	1	3	2	3	5	3.33
4	2	3	1	3	1	1	1.67

Table 2.9.: A sequencing scenario where different sequences are explicitly defined, i.e., the choice task is 'reduced' to a ranking scenario. In this example, sequence 1 has the highest *average* (AVG) value, i.e., it will be recommended first.

2.11. Polls and Questionnaires

A *poll* is a kind of sampling of opinions on a specific subject which is collected from a selected or a randomized group of persons. A *micro-poll* is a technical term for a short poll that is added, for example, to a website. Polls are used in situations where one is interested in the feedback of a group or a community with regard to a specific topic or question. Thus, polls are used to collect feedback which can be related to a decision, though the group asked is not necessarily affected by the result. Typical examples of such polls are 'how did you like the new version of our software?' or 'which version of the software do you use, the Android or the iOS-based implementation?'. Users participating in polls can be allowed to select one or more alternatives. A poll on the selection of the employee of the year could allow only one voting per user whereas a poll related to the selection of the best performer of a casting show could allow more than one vote. Systems supporting polls do not include any type of group recommendation functionality, in terms of supporting users in their decision making process. The aggregation mechanism applied in the context of polls is used to summarize the feedback of users (*ADD*-based aggregation) in terms of relative percentages per alternative (e.g., number of persons who voted for a candidate; see example in Table 2.10). In contrast to polls, *questionnaires* often consist of a collection of questions where the answer type of the questions can be defined in a flexible fashion (e.g., free text answers, multiple-choice answers, and single-choice answers). In some cases, questionnaires are defined on the basis of decision trees that specify in which context a question should be posed.

	u_1	u_2	u_3	feedback (ADD)
$q_1(1,2)$	1	1	1	1 (100%)
$q_2(1,2,3)$	2	3	2	2 (67%) 3(33%)
$q_3(1,2)$	1	1	2	1 (67%) 2(33%)
$q_4(1,2,3)$	1	2	3	1 (33.3%) 2(33.3%) 3(33.3%)

Table 2.10.: Evaluation scheme of polls and questionnaires – persons providing feedback often do not participate in the related decision making process.

2.12. Voting

Voting has a structure that is similar to polls, however, there is a decision aspect in voting since a group or a community decides on which alternative should be chosen, i.e., there is a clear pragmatics of the decision outcome. Typical examples of the application of voting are the *player of the month* (e.g., in soccer), the *reporter of the year*, and the *president of a country*. In many cases, the goal of voting is to select one alternative (e.g., the president), however, there are also scenarios where more than one alternative is selected. For example, in the context of a best paper award: if majority voting is used for determining a best paper and there is a tie (depending on the process) multiple alternatives could be selected as best papers. In the context of elections, the determined ranking of the alternatives has clear pragmatics. For example, the identified person becomes the new president. Elections can be single shot or iterative and different tie-breaking rules can be applied (an example thereof can also be a new election). An example of a voting process is shown in Table 2.11.

	u_1	u_2	u_3	result (ADD)
$a_1(0,1)$	1	0	1	$2\sqrt{\quad}$
$a_2(0,1)$	0	1	0	1
$a_3(0,1)$	0	0	0	0
$a_4(0,1)$	0	0	0	0

Table 2.11.: A voting process. Each user is allowed to give only one vote – a decision is made on the basis of the ADD aggregation function.

2.13. Further Aspects of Choice Scenarios

Tie-Breaking. Rules can help in situations where there is no clear winner but a decision has to be made. A tie-breaking *method* could be selected before the decision making process starts. This is used in situations where all group members agree on the method (or the method has to be accepted 'per-se'). Elections are an example of a situation where a group (in this case, a community) has to decide, and the method is already pre-defined. Further related examples are voting procedures in (public) organizations and companies, for example, when selecting a new rector for a university, selecting a new pope, or selecting the new president of the labor union. Situations where groups try to determine the tie-breaking method ahead of time also occur in less business-related decision processes. For example, what is the impact (weight) of the expert jury compared to the opinion of the audience collected via SMS votes in a TV show (in a situation where the jury ranking combined with the ranking of the audience does not result in a clear winner). Similar situations occur when it comes to the selection of the best paper at a conference – example resolution strategies in this context can be a simple majority-rules vote or the average rating the paper received from the reviewers.

Further examples of tie-breaking rules are *toss a coin* (useful, for example, in the context of low-

involvement items such as restaurants), *least misery* (useful in situations where two or more high-involvement alternatives have the same evaluation), *authority voting* (if a group did not agree on a specific decision rule and accepts the decision of a single authority), and *fairness* (in the context of repetitive decisions, users who were treated less favorably in previous decisions have priority). In many situations, a formalized and pre-defined rule for making a final decision does not exist, but the final decision is made on the basis of an internal discussion. In the 'best paper' scenario this means that the members of the jury simply analyze all the given alternatives and articulate their preferences, for example, in terms of an initial ranking. Given that every jury member has defined his / her preferences, a discussion can be started with the overall goal of achieving consensus between the group members. Such group decision-making requires the inclusion of forums which allow the discussion and exchange of views regarding (dis)advantages of alternatives (Nguyen and Ricci, 2017).

Multi-stage Processes. Multi-stage choice is performed if the decision making task can be separated into multiple phases (e.g., first decide about the date of the holidays and then decide on the location and the hotel), or the process itself may consist of the phase of identifying a consideration set (a set of candidate items that could potentially be chosen) and then selecting items from the identified consideration set. Examples thereof are personnel selections where the relevant candidates are pre-selected and – on the basis of the consideration set – hiring interviews are conducted. Further related examples are *idea management* (e.g., the selection of a name for a new product or the selection of topics that should be chosen for the next project proposal) or *strategic planning* (e.g., the definition and selection of new topics for professorships to be announced as open positions in the upcoming years).

Process Iterations. Iterative decisions (in contrast to *single-shot* decisions) are typically made in the context of high-involvement items, i.e., items with a higher negative impact triggered by a sub-optimal decision (compared to low-involvement items). In the context of such decisions, different types of conversational recommendation approaches, such as constraint-based recommendation and critiquing-based recommendation, are useful (Felfernig and Burke, 2008). Decisions related to high-involvement items are typically made in an *iterative* fashion, i.e., before the decision is made, a couple of iterations in terms of evaluations and discussions are performed. Examples thereof are manifold. For instance, a family purchases a new car, a new CEO is hired for a company, a group of students selects a new shared apartment, or a new ERP system is purchased by a company. Gamification-based approaches are a special case of iterative decision making, for example, *planning poker* (Haugen, 2006) is a consensus- and gamification-based approach to effort estimation (often used in requirements engineering; see Felfernig et al., 2017) where group members play cards. Each member holds a full deck of cards where each card represents a time effort ascending from, for example, 5 minutes to one month. After each group member has played a card (face-down), these cards are disclosed and the estimates of individual group members are discussed. After the discussion, each member plays another card until consensus is achieved. Examples of single-shot decisions are the selection of a

restaurant and the selection of a movie to be watched on the weekend.

Degree of Participation. Active participation is given if the persons providing preference feedback on the choice options are also engaged in the corresponding choice process (see also Felfernig et al., 2018). This is the case with most of the aforementioned scenarios, i.e., decision makers are also engaged in the feedback process and provide their preferences with regard to the given set of alternatives. The exception to the rule are *polls* and *questionnaires*, where communities provide feedback to decision makers but often do not actively participate in the decision making process.

2.14. Conclusions and Research Issues

In this chapter, we discussed choice scenarios that lie outside the scope of basic recommendations. We introduced a categorization of these scenarios along the dimensions of knowledge representation (items vs. parameters) and the inclusion of constraints. For a more in-depth understanding of these scenarios, we provided a couple of examples that show how to determine group recommendations. A couple of research issues also exist in this context. For example, the overall idea of group-based configuration is to engage user groups in configuration processes for complex products and services (Felfernig et al., 2016). Examples of such scenarios are the group-based configuration of software release plans, the configuration of smart homes, and the configuration of holiday packages. In all of these scenarios, approaches are required that support solution search that takes into account the preferences of individual group members. A specific issue is how to guide heuristic search when confronted with the preferences of a group of users. Initial related work can be found, for example, in Polat-Erdeniz et al. (2017). Similar aspects play a role when supporting groups in achieving consensus in the case of contradicting preferences. The research issue to be solved is how to include social choice mechanisms into preference elicitation, and corresponding diagnosis and repair processes. Initial work on the inclusion of personalization into diagnosis processes is presented, for example, in Felfernig et al. (2009, 2013).

Recommender Systems in Requirements Engineering

This chapter is based on the contents presented in Samer et al. (2021). The author of this thesis provided major parts of this chapter in terms of writing and literature research.

3.1. Abstract

Requirements engineering (RE) can be considered as one of the most critical phases in a software project. One of the main reasons of project failure is incomplete or missing RE. Intelligent approaches are needed to support stakeholders in the RE process to minimize the risk of project failure. In particular, there exists a high demand for intelligent tools such as *recommender systems* to increase the decision quality in the RE process, as the RE process is mainly driven by decision-making. In this article, we discuss a variety of recommendation approaches which have been proven to work well in the area of RE. Moreover, we depict how the application of modern concepts (e.g., gamification approaches or machine learning) can reshape the way of future RE. Extending beyond this scope, the article also provides an overview of tool support in RE and RE platforms which apply these technologies.

3.2. Introduction

Nowadays, technical systems and products play an important role in our society. Often, these systems are very feature-rich and complex. In particular, large software systems are usually planned, designed, managed, and developed in a project. Systems engineering usually starts with the investigation of the users' needs, which is known as *requirements engineering* (RE) today. After the analysis of the requirements, an optimal architecture is selected. This procedure is then followed by the development, integration, and evaluation of the system. RE plays a central role for the success of such projects (Hofmann and Lehner, 2001). In general, RE can be considered as a discipline which deals

with the elicitation and management of requirements. From an abstract point of view, a requirement describes what a customer or user expects from a (software) product in terms of properties, conditions, goals, and benefits (Ebert, 2014). According to Mobasher and Cleland-Huang (2011), requirements specify the functionality, constraints, and behavior of the proposed system. More formally, a requirement can be classified as (1) a *functional* requirement corresponding to a certain software feature, use case, or any other piece of functionality that should be included in the software product, or (2) a *non-functional* requirement which often refers to aspects such as reliability, correctness, performance, usability, profit, privacy, or software security. The whole process in which these requirements are defined and managed is called the *requirements engineering process*. Typically, this process is composed of different phases (also known as the *core activities*) which are often overlapping (see Table 3.6). These are the *definition and elicitation of requirements*, the *analysis and negotiation of requirements*, *quality assurance* (including the validation and management of the requirements), and *release planning*.

As mentioned, RE is crucial for the success of a software project. This is due to the reason that incomplete RE processes mostly lead to project failures (Felfernig et al., 2017; Hofmann and Lehner, 2001; Mobasher and Cleland-Huang, 2011). For example, Davis (2005) reports that 40% of project failures are caused by poorly defined requirements. Moreover, Krasner (2018) points out that up to 80% of the issues that lead to customer dissatisfaction regarding the delivered software originate from poor RE. According to Fucci et al. (2018), requirements are the third source of product defects and the main source of defects for service projects. The cost for fixing defects in the production phase can range up to huge extra costs that cannot be financed with the available project budget, whereas fixing defects in the requirements phase is much cheaper in comparison. Furthermore, Bokhari and Siddiqui (2010) point out that the specification of a requirement must be complete, correct, consistent, unambiguous, verifiable, and traceable. In order to reliably prevent possible project failures, each core activity / phase of the RE process has to be tested as well as carefully reviewed by RE experts. Even though requirements engineering (if done correctly) can prevent project failures and can also avoid defects in the final software product, only 2-4% of project resources are spent on it in many of today's software projects (Ninaus, 2016). A good design of the software as well as a clear and complete definition of the functionality a software system must provide, can eliminate potential sources of errors in the release planning phase and lower the risk of project failure. This is especially true for very complex software projects which is the default case in practice today (Damasiotis et al., 2017). The number of requirements a software project has, depends on the complexity of the project and can range from a few to up to many thousands (in case of a typical industrial software project). The increasing amount of functionality a typical software system must provide nowadays, leads to an overwhelming number of requirements which further increases the size and the complexity of the software and thus makes the RE process more challenging for engineers. Another problem is that a stakeholder who defines and describes a requirement is usually not the same stakeholder (e.g.,

developer) who will implement it later on. This leads back to the aforementioned issue of having incompletely and inaccurately defined requirements (Fucci et al., 2018). Furthermore, dependencies between requirements could exist which also have to be found and considered before release planning can take place. Unless all dependencies are found, planned releases may be in danger of not containing all of the planned features. Also, the identification of the right stakeholders responsible for a requirement appears to be a complex and challenging task in case of large software projects with hundreds or thousands of requirements. In other words, no single stakeholder can have the complete domain and background knowledge necessary to permanently make wise / smart and well-thought decisions. This is not only due to the fact that large projects consist of a large number of requirements but also due to the reason that these requirements often cover a broad variety of topics and different aspects of the project. This enormous amount of domain / background knowledge usually exceeds the level of knowledge a human stakeholder can possess. Hence, this has a negative impact on the quality of the decisions taken by the stakeholders in nearly all phases of the requirements engineering process.

The complexity of today's software projects as well as the fundamental importance of the decision quality of RE-related decisions, give rise to the application of intelligent methods in RE (Mobasher and Cleland-Huang, 2011). The mentioned complexity necessitates assistance in RE decision-making, to counteract situations where stakeholders can quickly become bogged down in a morass of details and find themselves spending a disproportionate amount of time seeking for information at the high expense of working on more productive tasks. To that end, *recommender systems* (RS), which are decision support systems, can be used to support RE-related decisions and to provide helpful assistance to stakeholders. Beyond the improvement of RE by using RS, the application of such systems has the potential to update current software engineering methodologies and to introduce new roles in software organizations (Samer et al., 2020). As a concrete example of state-of-the-art research in the field of RE conducted in recent years, we will shed some light on the European research project OPENREQ. In the scope of OPENREQ, research has been conducted in the field of RE with a focus on the application of intelligent recommendation technologies. OPENREQ offers intelligent recommendation tools and solutions for many RE-related problems. Samer et al. (2020) present the major outcomes of the project which touch a broad set of different communities, and the authors provide detailed descriptions and evaluations of the tools developed within the scope of the project. OPENREQ provides a variety of novel contributions in the field of RE, a good orientation for the scientific community, and solid foundations for the open-source community. As part of this overview article, we briefly highlight the major outcomes of the project that are most relevant for RE.

The major contributions of this overview article are threefold. First, we give an overview of current research related to the use of RS in RE. Second, we explain and compare different recommendation techniques that are of particular interest for typical RE scenarios. Furthermore, we also underpin the importance of RS for RE by providing basic RE-related examples that help to develop a better under-

standing of the presented recommendation techniques. Third, we provide new ideas regarding future work on how to improve the proactive support of users in RE by refining existing as well as exploiting new recommendation techniques.

In this overview, we explain basic recommendation technologies for RE, discuss the benefits of RS in RE for stakeholders, and provide a brief outlook on what they might do in the near future. Although RE is an engineering discipline that encompasses many different areas and domains such as traditional technical projects (e.g., in domains such as railway, telecommunication, etc.), it is necessary to limit and narrow down the scope to software projects, in order to orient the discussion topics of this chapter. The work presented in this chapter describes and analyzes current methodologies for (1) stakeholder and user involvement in the software life-cycle, (2) intelligent requirements engineering, and (3) major outcomes / results of central studies conducted within the scope of the OPENREQ project. The article summarizes related work regarding these techniques and thereby helps to develop a solid understanding of the area.

The remainder of this chapter is structured as follows. Section 3.3 explains the research methodology used to evaluate and determine relevant articles and publications from different literature databases. Section 3.4 presents related work with respect to the general application and use of RS in the context of RE. In Section 3.5, we describe the use of different recommendation approaches in various phases of the RE process. Section 3.6 provides a practical view of typical application scenarios of recommendation technologies. Additionally, some examples of RE tools which aim to support the stakeholders in this process, are presented. In Section 3.7, we discuss different selection criteria that allow to select appropriate recommendation solutions to be applied in various application contexts. Finally, at the end of this chapter, we recap our findings and give a summary of future research topics in RE (see Section 3.8 and 3.9).

3.3. Research Methodology

In order to provide decent insight into the current literature of existing recommendation technologies in the requirements engineering domain, we conducted a systematic analysis following the guidelines presented by Kitchenham and Charters (Kitchenham and Charters, 2007). At the beginning, we identified some key search terms that directly correspond to our topic. Examples of such search terms were “*recommender systems*”, “*recommendation techniques*”, “*intelligent approaches*”, “*intelligent systems*”, “*requirements engineering*”, and “*software engineering*”. Based on these terms, we gathered relevant papers from six well-known scientific database libraries including *Springer Link*¹, *IEEE*

¹Springer: <https://link.springer.com>

*Xplore Library*², *ACM Digital Library*³, *Research Gate*⁴, *Science Direct*⁵, and *Google Scholars*⁶. In order to further extend our review pool of paper candidates, we also searched for further papers in the proceedings of the most influential conferences related to software development and RE (see below). We reviewed the title, the abstract, and the conclusion of these papers in order to select potential candidates to be further analyzed and reviewed. Moreover, we selected our candidates according to the following criteria:

- Studies from prestigious journals, conferences, and workshops, such as conferences on *ACM Recommender Systems (RecSys)*, *ACM World Wide Web (WWW)*, *ACM User Modeling, Adaptation and Personalization (UMAP)*, *IEEE International Requirements Engineering Conference (RE)*, *IEEE/ACM International Conference on Software Engineering (ICSE)*, *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, *International Working Conference on Requirements Engineering (REFSQ)*, *ACM International Systems and Software Product Line Conference (SPLC)*.
- Only papers that have been published in English and after the year 2000.
- Significant impact in terms of relevance for the RE community as well as the range of influence.

In total, we collected 132 papers that meet all of the mentioned criteria. Among these candidates, we selected our final candidates based on their content using the following specifications:

- The work presented in this chapter clearly focuses on the topic *recommender systems* or *intelligent techniques* in RE.
- The authors' work presents clear objectives regarding the domain of interest.
- The paper provides interesting findings or conducted studies that show confident tests and good experimental or evaluation results.

This way, we identified 90 papers (out of the 132 papers) which are of high relevance and show a strong relationship to the discussion topics of this overview article. The most relevant studies and contributions of the reviewed papers have been cited or will be discussed in detail in the following sections of this chapter. Finally, we want to mention that the main contribution of this chapter is to not only provide a review report, but also a means to classify existing works into different categories based on taxonomical criteria presented in the remainder of this chapter (see Section 3.6).

²IEEE Xplore Library: <https://ieeexplore.ieee.org>

³ACM Digital Library: <https://dl.acm.org>

⁴Research Gate: <https://www.researchgate.net>

⁵Science Direct: <https://www.sciencedirect.com>

⁶Google Scholars: <https://scholar.google.at>

3.4. Related Work

In order to tackle the challenges described in Section 3.2, a lot of research has been conducted with respect to the aspects explained in Section 3.2. Different works identify a need for intelligent tool support to help requirements engineers and stakeholders in the different stages of the RE process (Castro-Herrera et al., 2009; Mobasher and Cleland-Huang, 2011; Ninaus et al., 2014). In particular, Mobasher and Cleland-Huang (2011) describe common RE tasks that require decision support and outline a variety of recommendation concepts applicable to solve different RE-related problems. Moreover, Hariri et al. (2014) discuss potential uses of RS in RE. The authors describe how RS can be used to extract requirements (in terms of feature requests) from discussions in online forums. Hariri et al. (2014) show that RS are suitable tools to discover new useful features in online discussions which can be included in new (software) products. The authors' work also sheds light on further application scenarios for RS in this particular context. For example, RS can propose experts among a stakeholder group for each forum topic and recommend relevant topics to all stakeholders. Earlier work of Castro-Herrera et al. (2008) also focuses on similar recommendation scenarios where stakeholders are assigned to different forums where they have to define formal requirements and work together in a collaborative fashion. In contrast to Hariri et al. (2014), the work of Castro-Herrera et al. (2008) deals with large-scale software systems and presents combined approaches of data mining and recommendation techniques to facilitate requirements elicitation in large-scale software projects.

Fucci et al. (2018) present more recent research of general application use cases for RS in RE. The case studies conducted by Fucci et al. (2018) identify the different areas which should be addressed by recommendation tools. The authors show that data analytics reduces information overload for individual stakeholders and help them to prioritize and even to find new requirements. To grasp the large amount of data, visualization support is needed for the decision-making process in the planning phase. In order to meet all of the given deadlines and to provide a better planning environment, tool support for time-wise project (and requirement) effort estimation is highly valued by companies. Moreover, RE tools should integrate with currently used upstream tools such as *Salesforce*⁷ to better identify customer needs.

Even though we can observe a strong trend of using RS in the context of RE, the aforementioned articles and research works primarily focus on some specific RE tasks but do not provide a general coverage of the entire RE process as a whole. Palomares et al. (2018) describe a general approach called THE OPENREQ APPROACH that guides the reader on how this issue can be addressed, in order to provide continuous recommendation support throughout the different phases of the whole RE process. The approach described by Palomares et al. (2018) has been successfully applied in the context of the European research project OPENREQ by different partners from the industry (e.g., SIEMENS, WINDTRE, and QT) and represents an innovative strategy for further industrial partners to support

⁷Salesforce: <https://www.salesforce.com>

RE in different real-world settings. Apart from the work presented by Palomares et al. (2018), more earlier work from Felfernig et al. (2013) also outlines common scenarios for RS in RE and the authors give a brief overview of basic recommendation concepts that can be applied in the different phases of the RE process. However, it is important to point out that the articles of Palomares et al. (2018) and Felfernig et al. (2013) only give a very rough overview of RS in RE and cannot convey a detailed picture of this broad research field. In contrast, this chapter gives a more broad as well as a more detailed and up-to-date overview of state-of-the-art research in this particular research field and presents existing recommendation technologies as well as results and key findings of modern recommendation approaches that have been developed in recent years. Moreover, the article also discusses latest research issues that may be of interest for RE practitioners in general as well as researchers working in the field of RE.

As already mentioned in Section 3.2, recommender systems (described in Section 3.5.1) are an appropriate solution to support the decision-driven RE discipline. A clear research agenda for recommender systems in the context of RE was defined by Maalej and Thurimella (2009). As part of their work, the authors provide a guideline that explains different approaches recommendation technologies should follow to assist stakeholders in the RE process. These approaches aim to foster a more efficient identification, analysis, and management of the requirements by exploiting important characteristics from the requirement data.

The fact that most RE tasks can only be completed by humans, complicates the usage of intelligent techniques. Consequently, the definition and elicitation of requirements, the negotiation and implementation of requirements as well as the quality assurance are still responsibilities for humans, no matter which technique was chosen. However, there are still remaining phases of the RE process where support by an intelligent technique may eliminate potential sources of errors. A recommendation approach, for example, may be helpful to suggest requirements to a stakeholder, who already dealt with the same topic in the past. As for finding dependencies between requirements, *natural language processing* (NLP) techniques combined with *artificial intelligence* algorithms can be used. Moreover, the creation of a release plan can be considered as a configuration problem. Although there already exist several individual tools for the different tasks in the RE process, the unique nature of the process makes it rather inconvenient and complex to combine these tools. INTELLIREQ (Ninaus et al., 2014) (see Section 3.6.2) addresses this problem and provides a web-based user interface that offers central access to an intelligent RE recommendation tool suite. Thereby, INTELLIREQ provides recommendation tools that allow stakeholders to save time and effort by reusing requirements and unveiling relationships (dependencies) between various requirements. The recommendation concepts developed for INTELLIREQ are either based on heuristic approaches or use basic content-based and collaborative filtering methods.

Similar to INTELLIREQ, the European research project OPENREQ (Samer et al., 2020; Palomares et al., 2018) also aims to tackle the aforementioned challenge by providing intelligent tools, approaches, and techniques for different RE scenarios to the RE community. In contrast to INTELLIREQ, OPENREQ’s tool suite covers a much broader range of recommendation functionality to support stakeholders in different RE tasks and aims to update current software engineering methodologies. Furthermore, the work conducted in OPENREQ focuses on systems that go beyond the basic heuristic rules or search mechanisms provided by INTELLIREQ. The developed systems use more sophisticated recommendation techniques which are mostly based on traditional as well as modern machine-learning techniques. As main part of this research project, a requirements engineering platform called OPENREQ!LIVE⁸ (see Section 3.6.2 and Samer et al., 2020) has been developed. Similar to the RE platform INTELLIREQ, OPENREQ!LIVE is a free web-based RE platform that allows users to jointly work on RE projects. The development of OPENREQ!LIVE was inspired by INTELLIREQ and follows the idea to offer a user-friendly access point to OPENREQ’s recommendation tool suite. The suite includes recommendation approaches to help stakeholders (1) in the definition of requirements, (2) in the identification of stakeholders responsible for the evaluation and implementation of a requirement, (3) in the improvement of the quality of the defined requirements, (4) in the prioritization and evaluation of requirements, and (5) during release planning. The OPENREQ!LIVE platform allows stakeholders to take advantage of the full recommendation power provided by the complete tool suite and fosters the cross-fertilization of ideas between stakeholders. On the platform, stakeholders find all necessary functionality to create and maintain a requirements model. In OPENREQ!LIVE, a requirements model consists of releases, requirements, dependencies (between the requirements), and release-related constraints such as the maximum capacity of a release (limiting the maximum number of requirements which can be included in the release) or the release’s deadline (date until when the implementation of the release must be complete, i.e., all requirements of the release are implemented). OPENREQ’s developed recommendation approaches constitute a fundamental basis of recent research in the field of recommender systems to enhance *software requirements engineering*. The results of OPENREQ as well as of international research conducted beyond this project represent innovative achievements in the RE community which are also presented throughout the remainder of this chapter.

3.5. Recommendation Technologies in Requirements Engineering

The rise of RS over the last decade, has led to an extensive development of different recommendation techniques. RS have also been successfully applied in the RE domain to help stakeholders to solve common tasks in the different phases of the RE process. In the field of computer science, RS are a very broad and popular research area. Many international conferences (e.g., ACM RecSys⁹) focusing on RS take place every year which receive broad attention as well as support from large companies and IT giants, such as Google, Amazon, Alibaba, Apple, Huawei, Facebook, or Microsoft. According

⁸OPENREQ!LIVE: <https://github.com/OpenReqEU/openreq-live>

⁹The ACM Conference Series on Recommender Systems: <https://recsys.acm.org>

to Ricci et al. (2010), Mahmood and Ricci (2009), and Burke (2007), a RS can be considered as a combination of several different software techniques and software tools which suggests items to a user that are relevant for him or her. Such *items* represent kinds of objects, e.g., buyable goods, music albums, news articles, holiday trips, or any other arbitrary kind of an intangible product or service such as a financial service. Normally, a RS primarily focuses on recommending a specific type of item as explained by Ricci et al. (2010). Many RS often have to deal with a large number of items and a fast growing number of users. This also applies to the RE process where software projects typically consist of a large number of requirements (e.g., these could be the *items* in some recommendation scenarios) and also quite often many stakeholders (e.g., they could be the *users* who receive the recommendations in some recommendation scenarios) are involved in the process. As a consequence, RS are predestined to be used in the context of RE and are appropriate tools to improve the quality of RE processes.

In the remainder of this section, we present different basic recommendation techniques relevant and applicable to support common RE-related tasks. The recommendation techniques presented in this section are based on *single-user recommendation* concepts, such as *content-based filtering*, *collaborative filtering*, and *knowledge-based recommendation approaches*. Additionally, we also explain the concept of *group RS* (Masthoff, 2015; Felfernig et al., 2018) which focus on recommending items to a group of users. The main objective of the discussed recommendation techniques is to improve and optimize the quality as well as the efficiency in requirements elicitation and management. Apart from the aforementioned recommendation concepts, there also exists a variety of more specific recommendation approaches, such as *time-aware* (Campos et al., 2014), *attribute-aware* (Liu et al., 2019), *context-aware* (Haruna et al., 2017), or *demographic* (Wang et al., 2012) algorithms which, however, lie out of scope of this chapter and are of minor or limited relevance for RE (despite of some specific application scenarios). For a more detailed analysis of the presented techniques, we refer to Jannach et al. (2010).

3.5.1. Basic Recommendation Algorithms in Requirements Engineering

In order to develop a basic understanding of recommendation technologies in general, basic concepts including *content-based filtering* as well as *collaborative filtering*, are explained in this section. Furthermore, we also provide small examples to demonstrate how these concepts can be used in the context of RE. This background knowledge represents the basis for the remainder of this chapter.

Content-based Approaches

Content-based recommendation approaches are used for the recommendation of content-based items. Often, content-based items are text documents such as newspaper articles, publications, newsgroup messages, or web pages (Pazzani and Billsus, 1997; van Meteren and van Someren, 2000; Pazzani and Billsus, 2007). However, content-based systems can also be effectively applied in the context of other

application domains (e.g., songs, movies, travel destinations, persons, products, etc.) where items are characterized by attributes (e.g., keywords, tags, metadata, title, author, words of description, etc.).

The underlying idea of content-based recommendations is to suggest new (yet unseen) items to users that are similar to the ones they preferred in the past, i.e., the recommended items share a similar content with the items previously liked / preferred by the user. The algorithm automatically extracts relevant feature values from the content of the items (or from the existing meta-information / attributes attached to the items) the user has previously rated mainly positive or liked. Given this extracted information, a user profile is created by the algorithm. The user profile reflects the individual preferences of the user on a general level (Pazzani and Billsus, 2007) and represents a model explaining the user's preferences based on the extracted features. The user profile is used to find and recommend similar items to the user. The similarity between the user profile and the individual items can be calculated by using an appropriate similarity metric such as the *Sørensen Dice coefficient*. The *Sørensen Dice coefficient metric* is shown in Formula 3.1. In general, the formula computes the similarity of two sets by measuring the relation between the number of elements common to both sets and the sum of the number of elements in each individual set. In this formula, $keywords(u_a)$ represents the set of profile keywords of user u_a and $keywords(r_x)$ corresponds to the keyword set of requirement r_x .

$$sim(u_a, r_x) = \frac{2 * |keywords(u_a) \cap keywords(r_x)|}{|keywords(u_a)| + |keywords(r_x)|} \quad (3.1)$$

In RE, requirements are usually described using natural language and characterized by additional attributes, such as the effort to implement the requirement, the general type (*non-functional* vs. *functional*), or a more specific category (technical feature, performance requirement, financial requirement, etc.). Hence, requirements can be interpreted as text-documents with meta-data (the attributes). Due to this reason, content-based approaches represent (in many cases) the most appropriate technique for requirements-related recommendation tasks. For example, a content-based recommendation approach could support stakeholders during the initial phase of a project when new requirements are defined. Such a content-based approach could help to reuse requirements from past (or other active) projects in the current project by recommending the relevant requirements related to the current project. This helps to avoid the redefinition of requirements which have already been defined in other projects by reusing these requirements. Content-based RS can also be used to identify relevant requirements for a stakeholder which are similar to the requirements the stakeholder has already investigated in the past. In both of the mentioned examples, a content-based RS can help to mitigate the risk of overseeing important requirements which saves time, reduces costs, and keeps the quality of the development on a high level.

Table 3.1 depicts a basic example consisting of two software projects. In this example, project B represents a new project (the current project) which deals with the development of a web interface to

control a smart home infrastructure. This project is currently undergoing the requirements definition and elicitation phase and we assume that the stakeholders have already defined two requirements (1 and 2) to be included in project B. Project A is an already completed software project and is related to the implementation of an analytics software for a sports watch. For demonstration purposes, only a small set of example requirements for both projects is shown. In this scenario, a content-based recommender system could recommend requirement 2 and 5 from project A to be included in the current project B since both requirements are related to the description / content of requirement 1 in project B. Moreover, the recommender could also propose to include requirement 1 from project A in the current project as there also exists a relation between this one and the requirement 2 of project B.

Collaborative Filtering Approaches

Collaborative filtering represents the most popular recommendation approach. In contrast to content-based recommendation, collaborative filtering follows the approach to model the concept of word-of-mouth suggestions between like-minded users (Ekstrand et al., 2011; Goldberg et al., 1992). In fact, one can observe that people usually tend to base their decisions on recommendations from other persons they know and trust, apart from their own experiences in their everyday lives. The authors of *Tapestry* (Goldberg et al., 1992), which was the first online RS, coined the term *collaborative filtering* and defined it as a concept based on social collaboration between users. Based on the behavior of like-minded users, new (yet unseen) items are recommended to the active / current user. For example, assuming that two users rated similar items in a similar fashion in the past, a recommender system based on collaborative filtering would recommend new items to one user that the other user has already rated positively. In other words, the approach first tries to find users who share similar tastes with the current user by considering those users who prefer the same items as the current user, and then recommends other items that are also preferred by the similar users but have not been viewed by the current user so far.

Collaborative filtering uses the concept of a *rating matrix* R (see Definition 3.2), also known as *utility matrix* or *user-item matrix*, for describing the user preferences. R contains all user-item interactions (explicit or implicit ratings) and represents a $m \times n$ matrix, where m denotes the number of users and n represents the number of items. We define the item i as an element in the set of all items I ($i \in I$) and the user u as an element of the set of all users U ($u \in U$). The element $r_{u,i}$ of the $m \times n$ matrix R represents the rating of the item i by user u . Using collaborative filtering, the rating $r_{u,i}$ of item i for the current user u is estimated based on the ratings $r(u', i)$ assigned to the same item i by those users $u' \in U$ who are similar (in terms of their rating preferences) to the current user u .

The collaborative filtering algorithm is responsible for finding similar users and for the prediction of ratings. There exist two different types of collaborative filtering approaches which are *neighborhood-*

Project A (past project)	<i>Smartphone Analytics Software for a Sports Watch</i>
ID	Requirement Description
1	For data privacy and security reasons, the software requires user authentication. This includes the registration of new users, login and account management. Users should be able to reset their password and to change their username and password.
2	The application requires a central database. The data should be stored in the database. The database storage is used for saving collected information.
3	For evaluating recorded training data, an evaluation algorithm is required. The software requires the connection and the access to the clock's internal memory.
4	Based on the data on height, weight, body fat, age and gender, the software should be able to calculate the ideal BMI for a user.
5	The software should be able to automatically transfer, backup and synchronize data collected by the device once a wireless connection has been established.

Project B (current project)	<i>Web Interface to Control a Smart Home Infrastructure</i>
ID	Requirement Description
1	Using wireless communication, sensor values can be read and saved in the database.
2	The user should be able to access his personal data in a restricted way by taking into account data privacy issues.

Table 3.1.: Basic example to demonstrate the potential of content-based recommendation approaches that foster the reuse of requirements in RE. In this example, the requirements 1, 2, and 5 (highlighted in bold) of an existing project A are recommended to be included in a new project which is project B.

based approaches and *model-based approaches*¹⁰. *Neighborhood-based* (also known as *memory-based*) collaborative filtering approaches generate recommendations by heuristically analyzing the utility matrix. Neighborhood-based approaches aim to find similar users (or items) based on similar tastes (ratings) with the current user (or current item). The underlying idea is that if users share the

¹⁰To limit the scope of this chapter, our discussion focuses on the explanation of collaborative filtering on the basis of neighborhood-based approaches.

same interests in the past, they will also have similar tastes / interests in the future. Items consumed by similar users are then recommended to the current user. Examples of commonly used similarity metrics (in this context) are the *Cosine similarity* and the *Pearson correlation coefficient*. The *Cosine similarity* (see Formula 3.3) describes the angle between two vectors in Euclidean space. In our case, the two vectors are the user vectors (\vec{u} and \vec{v}) that consist of the item-ratings of user u and user v , respectively. I_u as well as I_v refer to the two sets of all items that have been rated by user u and v , respectively. The Pearson correlation measure (see Formula 3.4) is a statistical metric that expresses the linear correlation between two (statistical) variables which are, in this case, the two user-vectors \vec{u} and \vec{v} . In Formula 3.4, I_{uv} represents the set of items that have been rated by *both* users.

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{pmatrix} \quad (3.2)$$

$$s(\vec{u}, \vec{v}) = \cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \times \|\vec{v}\|} \quad (3.3)$$

$$s(\vec{u}, \vec{v}) = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)^2 \sum_{i \in I_{uv}} (r_{v,i} - \bar{r}_v)^2}} \quad (3.4)$$

Table 3.2 shows a basic example of a simplified requirements evaluation scenario. In this example, we demonstrate how collaborative filtering can be used to assist stakeholders in the finding of relevant requirements. For simplicity, we assume that a collection of 4 requirements is evaluated by 4 stakeholders using 5-star ratings. In this example, the ratings indicate the preference a stakeholder has for a certain requirement. Using standard collaborative filtering (i.e., user-based neighborhood collaborative filtering) and assuming that Bob is our current / active user, we would first try to find similar / neighbor users by comparing the similarity. In this example, Bob has only evaluated requirement 2 and requirement 3. Both requirements have also been evaluated by Ann and Chris, whereas Susan has only evaluated requirement 3 in common with Bob. Since Susan's rating of requirement 3 strongly differs from Bob's rating, she cannot be considered as a neighbor / similar user. However, Ann has rated both requirements in a quite similar fashion than Bob, and Chris has given the same ratings for both requirements as Bob. Hence, Ann and Chris can be considered as potential neighbors. According to Table 3.2, these potential neighbors (Ann and Chris) have also given a positive feedback for requirement 4. Consequently, the algorithm would recommend Bob to take a look at requirement 4.

Note that the ratings shown in Table 3.2 just represent a basic scenario where stakeholders provided explicit feedback to requirements in terms of ratings. However, in practice, such feedback could also

be implicitly derived from log data (e.g., the number of times a requirement has been viewed by the stakeholder) that expresses the degree of preference a stakeholder may have for a certain requirement (for more details on implicit feedback, we refer to (Hu et al., 2008)).

The main advantage of a collaborative-filtering-based approach over a content-based filtering approach is that this approach does not rely on the content of the requirements (textual description of the requirements, attachments, etc.) and can be applied once a sufficient amount of rating data exists. Moreover, content-based approaches do not support serendipity aspects (Kotkov et al., 2016) (see also Section 3.7). This means that only items / requirements can be recommended that share a similar content with the already rated items, but not items / requirements which are completely unrelated to the items / requirements that have been rated before. However, collaborative filtering is capable of providing serendipity-based recommendations which represents a significant advantage over standard content-based recommendation approaches. Hence, collaborative filtering helps to provide more diversity in the recommendations and empowers stakeholders to explore a broader set of potentially relevant requirements which are not related to each other on the level of their (descriptive) content.

Stakeholder	Requirement 1	Requirement 2	Requirement 3	Requirement 4
Ann	-	5	4	5
Chris	1	4	5	5
Susan	3	-	2	-
Bob	-	4	5	-

Table 3.2.: Basic example to demonstrate the potential of collaborative filtering approaches in RE. Ann and Chris have rated the same requirements (requirement 2 and 3) in a similar fashion as the active user Bob. Hence, Ann and Chris are considered as similar / neighbor users and requirement 4 (which is highlighted in bold and has been highly rated by them but has not been seen by Bob yet) is recommended to Bob.

3.5.2. Advanced Recommendation Algorithms in Requirements Engineering

In addition to the already presented concepts, more specific recommendation techniques are explained in the remainder of this section. These techniques represent approaches that are applicable in more complex recommendation scenarios. To demonstrate the functionality of these approaches for RE, we provide basic examples along with detailed explanations.

Knowledge-based Recommendation

In sharp contrast to the previously discussed approaches, *knowledge-based* systems (Felfernig et al., 2014) are usually used to recommend *high-involvement* or *complex items* such as digital cameras, financial services, or tourist destinations. In contrast to *high-involvement*, *low-involvement items* are

typically often consumed or bought items in the everyday life, which require little decision effort from the user. Examples thereof are consumer goods (such as books, mobile phones, etc.), consumer services (such as music songs, cinema movies, etc.), food (such as milk, bread, toothpicks, etc.), and any other kind of mass-produced goods (e.g., toys for children, kitchenware, etc.), for which there are no major differences in quality. When deciding for such items, the user does not have to deal with the alternatives in detail so much. Often such *low-involvement items* also have a low price and the consequences of a wrong decision taken by the user are very limited. With *high-involvement items*, the willingness to obtain information is much higher and often vital for the decision taken by the user. One example is the purchase of a new car, since cars are very costly (when compared to most *low-involvement items*) and often seen as status symbols. Further examples include the purchase of a new house, or important business decisions taken at large companies. In the case of *high-involvement items*, the consequences of a wrong decision can be very serious and sometimes even unforeseeable (e.g., a wrong impactful business decision could result in a situation where the company goes out of business). As already mentioned before, the focus of *knowledge-based* systems (Felfernig et al., 2014) lies on *high-involvement items*. Normally, for such items no sufficient and no up-to-date purchase data as well as rating data is available as these items are usually not purchased often. Thereby, the knowledge-based algorithm generates recommendations based on predefined recommendation settings / rules (including user preferences) as well as specific domain knowledge and meta-information about the item assortment.

Formally explained, a RS (in general) can be interpreted as *any system which is used in a large space of possible options to guide a user in a personalized fashion to objects that are potentially interesting or useful for the user* (Burke et al., 2011; Ninaus et al., 2014). However, a RS in terms of a knowledge-based recommender, can also be a system which produces such objects / solutions as main output / result (Burke, 2000). *Knowledge-based* RS are of special interest to be applied in the context of RE. In this context, a generated solution could be a release plan that satisfies a given set of constraints, such as interdependencies between the requirements, the maximum amount of requirements that can be assigned to each release, or the release deadline until when all requirements of the release have to be implemented.

Table 3.3 demonstrates an example scenario for the generation of a release plan. In this simplified example, a release plan solution of a software project is determined using a knowledge-based RS. The knowledge-based RS works as a configuration system and receives a requirements model as input (see left column). The requirements model consists of 4 requirements, 2 releases, and 3 constraints. One constraint "Requirement 2 \rightarrow Requirement 4" describes an interdependency relationship between requirement 2 and requirement 4. In this case, the notation "Requirement 2 \rightarrow Requirement 4" expresses that requirement 2 cannot be implemented before requirement 4 has been implemented. In order to satisfy this constraint, a release plan must assure that requirement 4 is part of an earlier release than

requirement 2. The effort of the implementation of a requirement is usually estimated by experts. In this example, the effort to solve each requirement is stated in working hours. The priorities of the requirements signify how relevant and urgent the requirements are for the project. Considering this information, a release plan should typically have all high-priority requirements in early releases and all lower prioritized requirements in later releases. Moreover, there exist two further release-related constraints in this model which define the maximum capacity of both releases in working hours (100h for release 1 and 150h for release 2). The size of a release's maximum capacity depends on different factors, such as the number of stakeholders in a project, their availability to work in the project, the date / dateline of the release, etc. The maximum capacity of a release strictly limits the maximum working effort (in hours) that can be spent by the stakeholders to work on the release such that the deadline of the release can be fulfilled. Depending on this constraint, the decision which requirements and how many requirements (e.g., many low effort requirements vs. a few high effort requirements) can be assigned to the release is limited.

A release plan solution determined by a knowledge-based configuration system for the given requirements model is shown on the right side of Table 3.3. The proposed release plan consists of 2 releases containing 2 requirements that satisfy all 3 constraints. The release plan guarantees that the implementation of requirement 4 is complete before requirement 2 gets implemented. Furthermore, the maximum capacities of both releases are not exceeded by the proposed requirement assignment (effort to implement release 1: 50h [Req. 4] + 30h [Req. 1] = 80h; effort to implement release 2: 90h [Req. 2] + 40h [Req. 3] = 130h). Finally, the proposed release plan also aims to satisfy the individual priorities of the requirements as best as possible (the high-priority [Req. 4] and one middle-priority requirement [Req. 1] are part of the earlier release).

Input:	Output (proposed release plan):
<i>Requirements:</i>	<i>Release 1 (Total effort: 80h)</i>
- Req. 1 (Effort: 30h, Priority: MIDDLE)	- Req. 4 (Effort: 50h, Priority: HIGH)
- Req. 2 (Effort: 90h, Priority: MIDDLE)	- Req. 1 (Effort: 30h, Priority: MIDDLE)
- Req. 3 (Effort: 40h, Priority: LOW)	<i>Release 2 (Total effort: 130h)</i>
- Req. 4 (Effort: 50h, Priority: HIGH)	- Req. 2 (Effort: 90h, Priority: MIDDLE)
<i>Dependencies:</i>	- Req. 3 (Effort: 40h, Priority: LOW)
- Req. 2 → Req. 4	
<i>Releases:</i>	
- Release 1 (Maximum Capacity: 100h)	
- Release 2 (Maximum Capacity: 150h)	

Table 3.3.: Basic example of a recommended release plan (requirements are abbreviated as "Req.") considering requirement- and release-related constraints such as requirement effort, requirement priority, interdependencies between the requirements, and the maximum capacity of the releases

Group Recommendation Approaches

In general, there exist two different types of group recommendation approaches. The first type are group recommender systems that extend the basic concept of single-user recommender systems and are able to discover new (yet) unseen items. Basically, this type of group recommender systems can either (a) utilize basic single-user recommendation algorithms to combine the (single-user) recommendations for each group member to a single recommendation list presented to the group, or (b) utilize these single-user recommendation algorithms to generate group recommendations given a user profile representing the preferences of all group members (instead of a single user). In both cases, already discussed (see also Section 3.5.1) single-user recommendation algorithms (such as content-based or collaborative filtering algorithms) are applied. One application scenario of such systems can be the recommendation of a new holiday destination to a tourist group. However, such systems are usually of limited interest for RE since there hardly exists any use-case in RE where a group of users / stakeholders wants to jointly discover new items. Due to this reason, this type of group recommender systems is not further discussed in this section and for more details we refer to Felfernig et al. (2018).

The second type of group recommender systems are systems that do not recommend new (yet) unseen items to a user, but provide a prioritization of already seen / known items to a group of users. These kind of group recommenders are applied in scenarios where a group of users has to decide for and choose one item among a list of different candidate items (e.g., different family members who decide on the travel destination for their next vacation trip among different candidate destinations). In order to generate a prioritization, the system requires from each group member to rate all candidate items. The focus of such group recommender systems does not lie on the discovery of new items for a user, but on deciding for an item among a set of known and rated items. The purpose of such group recommender systems is to generate and propose a prioritized list of items to the user, in order to help the user in making the decision of selecting the final candidate. In this context, the recommendations are often made using *group decision heuristics* (Masthoff, 2015; Felfernig et al., 2018). For example, the heuristic *least misery* tries to find recommendable items for the group by ignoring as much as possible those items that are disliked the most by the group members (see Formula 3.5). In contrast to that, *most pleasure* (see Formula 3.6) is another heuristic that recommends the item with the highest of all individual ratings, but does not consider the number of group members who really like the item.

$$LMS = \arg \max_{(r \in I)} (\min (eval(r))) \quad (3.5)$$

$$MPL = \arg \max_{(r \in I)} (\max (eval(r))) \quad (3.6)$$

The concept of *multi-attribute utility theory* (MAUT) (Ninaus et al., 2014; Felfernig et al., 2018; Atas et al., 2018; Samer et al., 2020) represents an alternative to the aforementioned group decision heuristics. Utility-based recommendation approaches utilize the concept of combining multiple attributes in a linear fashion. The attributes (also known as *interest dimensions*) of an item are rated individually by each group member. In terms of RE, examples of such attributes are the potential profit (p) of an implemented requirement, the effort (e) related to the implementation of the requirement, or the risk (r) that the requirement cannot be implemented. Using the attribute-ratings provided by all group members for a requirement (item), MAUT-based group recommenders can determine the priority of the requirement by calculating its utility.

Formula 3.7 shows the basic calculation of a requirement's utility for a single user s using a linear combination of the user's attribute-ratings. This linear combination of the attribute-ratings can be interpreted as a weighted average of the attribute-ratings where the attributes (interest dimensions) have different weights. These weights are called *importance-weights*, since every *importance-weight* $imp(d)$ of an attribute d defines the degree of importance the attribute d has. The *importance-weights* can either be learned from historic data or manually defined by experts. In Formula 3.7, $eval(r, d)$ refers to the attribute-rating of the attribute d for requirement r provided by user s and $imp(d)$ corresponds to the importance of attribute d .

$$util(r, s) = \frac{\sum_{d \in D} eval(r, d) * imp(d)}{\sum_{d \in D} imp(d)} \quad (3.7)$$

Formula 3.8 extends Formula 3.7 and explains the concept of MAUT for a group of users. In this formula, the set of group members is defined as S . Each attribute-rating of every individual group member ($s \in S$) is also weighted by another weight factor which is called the *expertise weight* $w(s, d)$ and represents the skill-level of a stakeholder s for attribute d . Similar to the *dimension-weights*, all *expertise weights* can either be learned from historic data or manually defined by the project managers. In order to determine a utility value for the requirement r (i.e., $util(r)$), the weighted sum of all attribute-ratings provided by the group members is divided by the number of group members $|S|$.

$$util(r) = \frac{\sum_{s \in S} \frac{\sum_{d \in D} eval(r, s, d) * imp(d) * w(s, d)}{\sum_{d \in D} imp(d) * w(s, d)}}{|S|} \quad (3.8)$$

Table 3.4 presents an overview of a basic group-based requirements evaluation example. In this example, three stakeholders evaluate three requirements $\{r_1, r_2, r_3\}$ with regard to the three attributes *profit*, *effort*, and *risk*. The expertise of the three stakeholders (skill-level) $w(s, d)$ is defined on the level of the three interest dimensions (see Table 3.5). The values of the stakeholders' skill-levels lie

in the range between 0 and 1, and can either be defined by the stakeholders or automatically determined from previous evaluation activities. All attribute-ratings lie in the range between 1 and 5. In terms of *profit*, a rating value of 5 indicates that the requirement highly contributes to the profit of the project, whereas a rating value of 1 signifies that the requirement has a very low impact on the project's profit. Likewise, the attribute *risk* reflects how risky the non consideration of the requirement is for the project. A high rating value for the attribute *risk* expresses a higher risk for the project and vice versa. In contrast, a rating value of 1 for effort means that the effort related to the implementation of the requirement is very high, whereas a rating value of 5 corresponds to a minimal amount of implementation effort.

Stakeholder	r_1			r_2			r_3		
	profit	effort	risk	profit	effort	risk	profit	effort	risk
Susan	4	4	2	4	5	5	2	5	5
Patrick	2	5	4	5	4	5	2	5	4
Olivia	5	3	5	5	3	5	1	5	5
Utility value (priority)	4.05			4.71			3.83		
Priority ranking	2			1			3		

Table 3.4.: Three stakeholders ($S=\{Ann, Chris, Susan\}$) evaluated the requirements $\{r_1, r_2, r_3\}$ with regard to the attributes *profit*, *effort*, and *risk*. The calculation of the utility values is based on Formula 3.8. The priority ranking defines the suggested ranking of the requirements based on their determined utility.

In order to recommend a prioritization of the requirements to the group (in terms of a ranked list), the utility value for each requirement is calculated using Formula 3.8. For each requirement, we apply Formula 3.8 to the attribute-ratings shown in Table 3.4 by using the *expertise weights* from Table 3.5 and different *importance weights* for each attribute. In this example, we use the *importance weights* $imp("profit")=0.4$, $imp("effort")=0.3$, $imp("risk")=0.5$ which can either be learned from historic data or manually defined by experts (as already mentioned before). The calculated utility values for the requirements r_1, r_2, r_3 are shown in Table 3.4. After the calculation of the utility values for all requirements, the group recommender system suggests a list of prioritized requirements sorted by their utility values in reverse order (a high utility / priority corresponds to a low rank in the prioritized list and vice versa). By using this basic concept of requirements prioritization (based on the group member attribute-ratings), the group recommender system is able to take into account as much as possible the preferences of the whole group. In this example, the recommended sequence of prioritized requirements is $[r_2, r_1, r_3]$ (see last row of Table 3.4) which represents a recommendation that the three requirements should be implemented in this chronological order (r_2 , then r_1 and finally r_3).

Stakeholder	profit	effort	risk
Susan	0.2	0.8	0.1
Patrick	0.2	0.3	0.3
Olivia	0.5	0.3	0.4

Table 3.5.: Expertise level $w(s, d)$ of the individual stakeholders in the range between 0.0 (minimum) and 1.0 (maximum) with respect to the interest dimensions profit, effort, and risk.

Further Approaches and Extensions

Liquid Democracy. A more sophisticated approach which recently emerged in the RE community and aims to make the preference elicitation of the previously discussed group recommendation approaches more flexible, is based on the idea of *liquid democracy*. In the context of RE, *liquid democracy* is a vote-delegation concept that extends the existing group recommendation approaches by allowing users / stakeholders to transfer their votes to other experts. In general, group recommendation systems are more powerful tools to evaluate and prioritize requirements (than single-user recommenders) as they allow to combine the expertise of many stakeholders (group members). One reason for this is that stakeholders have a varying background knowledge and different viewpoints. Existing research shows that group-decision tasks (such as the evaluation and prioritization of requirements) can benefit from this variety of background knowledge and opinions (Johann and Maalej, 2015; Zhang and Zhou, 2017; Atas et al., 2018; Samer et al., 2020). However, not every stakeholder can have the full expertise to evaluate requirements with regard to all different aspects. In reference to the previously shown group recommendation example in Section 3.5.2 (see Tables 3.4 and 3.5), *liquid democracy* allows stakeholders to delegate their votes / ratings either for a certain attribute (interest dimension) or a requirement to another stakeholder. In particular, a vote transfer on the level of attributes can provide a highly qualitative benefit to the entire requirements evaluation process. For example, if there is a software developer called Olivia in a software project who does not feel confident to evaluate / rate requirements with regard to the attribute *profit*, then she can decide to delegate the vote for this certain attribute to the project manager Susan. Moreover, Susan can further decide to delegate all her votes for the attribute *effort* to a more technical person such as Olivia. The delegation of all attribute-votes for a certain requirement also represents another option towards a more fruitful evaluation of requirements in group-based environments. This allows stakeholders who cannot rate a requirement (for any reason), to increase the voting power of an expert by delegating all his / her attribute-votes for the requirement to this expert instead of only abstaining from voting the requirement without delegating the attribute-votes to the expert.

The most effective way to consider attribute- or requirement-specific vote delegations in a simple manner, is by controlling the *expertise weights* of the recipient stakeholder who receives a vote. At the beginning of the requirements evaluation process, the *expertise weights* of all stakeholders (S) can be initialized based on the different expertise levels the stakeholders have for the corresponding attributes. As already mentioned in Section 3.5.2, these (initial) *expertise weights* can be learned

from historic data or manually defined by project managers. In the most simplest case, all initial *expertise weights* can also be set to 1 for all stakeholders and solely be adapted / learned by using *liquid democracy*. Regardless of the initialization, the procedure of expertise weight adaptations remains the same. Whenever a stakeholder s transfers a vote to stakeholder s' for a certain attribute d , then the *expertise weight* $w(s', d)$ of s' is incremented by the value $w(s, d)$ which is the *expertise weight* of stakeholder s for attribute d . After that, the stakeholder s is no longer allowed by the system to evaluate requirements with regard to attribute d . Moreover, stakeholder s' who received a vote from stakeholder s for a certain attribute d can decide to further delegate the vote (now including the vote of s and s') to another stakeholder. This delegation process can be repeated any number of times by the next recipient and all delegations can be visualized in a directed graph.

Figure 3.1 presents a basic example of such a vote-delegation graph showing an entire chain of vote delegations. Each node in this graph corresponds to a stakeholder and every directed edge between two stakeholder-nodes represents a delegated vote. The graph depicts the complete vote-delegation hierarchy for a certain attribute (in case of delegations made on the level of attributes) or a certain requirement (in case of delegations made on the level of requirements). In order to allow the calculation of the *expertise weight* for the (last) recipient-stakeholder (s_1 in this example), the delegation mechanism must always ensure that no recipient-stakeholder (i.e., s_4 , s_2 , s_3 , and s_1) can delegate his / her vote *back* to any of the previous stakeholders in the graph. This guarantees that the vote-delegation graph will never have cycles and always keeps its (directed) acyclic graph structure. A simple mechanism that can detect undesired back-votes is based on the *depth-first search* algorithm (Tarjan, 1971). The mechanism must be triggered before a stakeholder delegates the vote and it checks whether the recipient-stakeholder (selected by the delegating person) already appears in the delegation-graph of the delegating person. Since every stakeholder can only delegate his / her vote once, this *directed acyclic graph* always represents a *directed tree* regardless of a stakeholder's delegation behavior. Given this directed acyclic vote-delegation tree structure, the calculation of the *expertise weight* is a trivial task where only the *expertise weight* of the last recipient stakeholder must be calculated. The *expertise weight* of the last recipient stakeholder is the sum of the initial *expertise weights* of all stakeholders who delegated the vote either directly or indirectly to this (last) recipient. In case of Figure 3.1, stakeholder s_1 is the (last) recipient. The *expertise weight* of s_1 is the sum of the initial weights of all stakeholders (stakeholders $s_2 - s_6$) who delegated the vote directly or indirectly to s_1 . Assuming that the initial weights of all stakeholders are 1 at the beginning, the expertise weight of s_1 is 6 (i.e., $w(s_1, d) = 6$).

The utility of the requirements and the prioritization (list of requirements ranked by their utility value) can be determined by using Formula 3.8, the attribute-ratings provided by the voters, and the adapted *expertise weights*. In case of requirement-specific vote delegations, the same procedure is applied, however not only for a single *expertise weight* but for all *expertise weights* of s . Moreover, in this case, all *expertise weights* of the recipient stakeholder s' are only updated for the corresponding requirement and not in general.

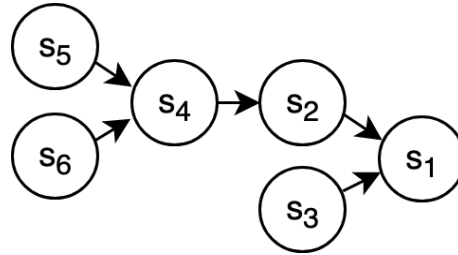


Figure 3.1.: Liquid democracy. Example of a vote-delegation tree for a single attribute or a single requirement, respectively. All vote-transfers are visualized as (directed) edges and all stakeholders are represented as nodes.

Supervised Classification. In many of today’s scenarios, recommender systems have to cope with a growing number of users and a growing amount of user activity. This leads to situations where the capacity limits of standard content-based and collaborative filtering algorithms are typically reached. For that reason, content-based and collaborative filtering algorithms have to utilize more efficient and powerful learning concepts such as supervised classification. In general, supervised classification belongs to the category of supervised machine learning approaches. Algorithms based on supervised classification can range from *traditional machine learning algorithms* (such as *Logistic Regression*, *Random Forest*, *Decision Tree*, *Support Vector Machines*, etc.) to *modern deep learning algorithms* which utilize the concept of deep neural networks. Regardless of the used algorithm, the algorithm builds a model that learns a function from given pairs of input samples and their corresponding output labels. The samples usually refer to entities such as items or objects (e.g., products, newspaper articles, movies, etc.). Each sample consists of *features* (characteristic information) and a *label* (the class). The features are (quantitative or qualitative) numeric values and can be mathematically described using a vector representation (the feature vector).

Formula 3.9 expresses the approach of classification in terms of a mathematical representation, where the function f_W is a parametrized function that refers to the learned prediction model with parameters W . Formally explained, the learned prediction model represents a mathematical function f_W which receives a multidimensional feature vector of a sample ($\vec{x} \in \mathbb{R}^D$) as input and returns a predicted label ($\hat{y} \in \{0, \dots, k-1\}$). The supervised learning algorithm should produce a function $f : \mathbb{R}^D \rightarrow \{0, \dots, k-1\}$ given the samples in terms of feature vectors ($\vec{x} \in \mathbb{R}^D$) and their corresponding labels ($y \in \{0, \dots, k-1\}$) such that the prediction model learns the association to which of the k categories (classes) each of the given samples belong to (i.e., $\hat{y} \approx y$). This is achieved by adapting the entries of the model parameter matrix $W \in \mathbb{R}^{I \times J}$ during training.

$$\hat{y} = f_W(\vec{x}) \quad (3.9)$$

The goal of supervised classification is that after several arithmetic operations with different inputs and outputs / labels, the model is trained to create associations between the samples and their labels. In order to work, supervised classification algorithms require a label assigned to every sample. Examples of supervised classification applications are diverse and include email spam detection, handwriting recognition, object or face recognition in images, and speech recognition.

In terms of RE, supervised classification is typically used for ambiguity detection in requirements to find ambiguous, unclear, or malformed descriptions in the text of the requirements. Another RE-related use case represents sentiment analysis, which can be used, for example, to automatically determine and prioritize more critical requirements rather than less critical ones. Moreover, classification can also be used to classify requirements based on their type (e.g., non-functional requirements related to project-specific details, requirements that describe software bugs, requirements that describe technical features, etc.).

As mentioned, the use of supervised classification requires the existence of labeled data samples. Sometimes, the labels are already available or they can be automatically collected together with the sample data in some real-world scenarios. However, in many practical scenarios, the samples have to be manually labeled by humans. The manual labelling of a large number of samples is often too expensive and not practical due to a limited amount of available human power, time, and budget resources. In such cases, it is a common way to apply unsupervised machine learning techniques such as *clustering*.

Clustering. Another recommendation technique is recommendation based on clustering. Clustering is a method which falls into the category of unsupervised machine learning approaches. In general, clustering is used to identify similarity structures in (usually relatively large) databases. In sharp contrast to supervised classification techniques which require *labeled* data samples that are assigned to existing classes, clustering does not depend on the previous knowledge of the classes to which the samples belong to. Clustering works with *unlabeled* data samples (representing items) and it groups together the similar samples based on the similarity of characteristic information (e.g., attributes, textual content, etc.) related to the items. Basically, clustering represents an agglomeration analysis of a set of different items / objects, where these items are compared based on a similarity score, and closely related (i.e., similar) items are grouped together. The groups of similar items that are found in this way are called *clusters* and the assignment of these groups is referred to as *clustering*. There exists a variety of different clustering algorithms which are based on different modelling approaches. Examples thereof include graph-theoretical, hierarchical, partitioning, fuzzy, or optimizing models (Xu and Tian, 2015). The numerous clustering algorithms primarily differ in terms of similarity, their cluster model, their algorithmic procedure, and the tolerance to strongly varying data. Under certain circumstances, a clustering algorithm can reproduce existing knowledge by finding groups that refer to known classes or categories (for example, divide personal data into the well-known groups "female" and "male"), or also generate groups that may not directly refer to known classes or categories.

In the latter case, the determined groups / clusters can often not be described verbally (e.g., "female persons") and common characteristics of the clusters can usually only be identified through a subsequent analysis. When using cluster analysis, it is often necessary to try different methods and different parameters, to preprocess the data and, for example, to select or omit some attributes / dimensions. The clusters determined using clustering can be used, for example, for market segmentation or to recognize patterns in images.

In the context of RE, software projects often consist of a large assortment of unlabeled requirements which belong to a certain category / type. Clustering-based approaches help requirements engineers to automatically group the unlabeled requirements based on their (textual) content, to different groups or requirement types for further evaluation. This allows stakeholders not only to find duplicates, dependencies (between requirements), and outliers more easily, but also to develop a better understanding on how the pool of requirements is structured (Cleland-Huang et al., 2009; Deshpande et al., 2019; Samer et al., 2019). Another use case of clustering in RE is to automatically group similar stakeholders for the evaluation of requirements and to recommend relevant requirements to these groups, which are related to the skills of these stakeholders. This way, clustering can be interpreted as a method that extends the set of the presented recommendation techniques.

3.6. Application Areas of Recommenders in Requirements Engineering

In this section, we discuss different recommendation approaches and tools which are related to traditional RE as well as to more agile and open-source oriented areas of application. These tools are based on the previously discussed recommendation technologies which have been presented in Section 3.5. Moreover, we present two representative RE platforms which combine the functionality of the selected set of existing tools and provide a single user interface for stakeholders to access these RE tools.

3.6.1. Recommenders in Traditional Requirements Engineering

Traditional RE (Nuseibeh and Easterbrook, 2000) follows a predefined procedure that can be referred to as the *RE process*. This procedure includes different phases / activities (see also Felfernig et al., 2013). In the literature, these activities are usually described as being independent and chronologically ordered. However, in practice, these activities are often iterative, repetitive, and interleaved phases that can be distributed throughout the entire development life cycle of a software product. This subsection provides an overview of recommendation approaches and their application areas in the RE process. Table 3.6 presents an overview of these recommendation approaches grouped by the different RE activities in which these approaches can typically be applied.

Definition And Elicitation of Requirements In the initial phase of the RE process, requirements, which are potentially relevant for the software project, are collected by the different stakeholders.

Activities	Paper references
I. Definition & Elicitation	Navarro-Almanza et al. (2017), Ivan et al. (2016), Kanchev et al. (2017b), Slankas and Williams (2013), Falkner et al. (2019), Winkler and Vogelsang (2016), Goldin and Berry (2015), Oriol et al. (2018), Lu and Liang (2017), Baker et al. (2019), Casamayor et al. (2010), Binkhonain and Zhao (2019), Khan et al. (2020), Dhinakaran et al. (2018), Abualhaija et al. (2019), Dalpiaz et al. (2019), Stanik and Maalej (2019), Kanchev et al. (2017a), Johann et al. (2017), Kurtanović and Maalej (2017), Khan et al. (2019), Stanik et al. (2019), Dekhtyar and Fong (2017), Groen et al. (2015), Guzman et al. (2017), Mahmoud (2015), Tizard et al. (2019), Abad et al. (2017), Williams and Mahmoud (2017), Dumitru et al. (2011)
II. Analysis & Negotiation	Castro-Herrera et al. (2008), Samer et al. (2018), Mobasher and Cleland-Huang (2011), Kifetew et al. (2017), Sadiq et al. (2017), Ninaus et al. (2014), Lim et al. (2010), Karlsson (1996), Castro-Herrera and Cleland-Huang (2010), Felfernig and Ninaus (2012), Samer et al. (2019), Ruhe et al. (2002), Samer et al. (2020), Felfernig et al. (2018), Perini et al. (2013), Duan et al. (2009), Stanik et al. (2018), Tonella et al. (2013)
III. Quality Assurance	Deshpande et al. (2019), Danylenko and Löwe (2012), Atas et al. (2018), Linsbauer et al. (2013), Tiwari and Laddha (2017), Parra et al. (2018), Fitzgerald et al. (2011), Mezghani et al. (2019), Ninaus et al. (2014), Mahmoud and Niu (2011), Felfernig et al. (2013), Ferrari and Esuli (2019), Ferrari et al. (2018), Ferrari and Gnesi (2012), Cleland-Huang et al. (2009), Felfernig et al. (2010), Gleich et al. (2010), Samer et al. (2019), Linsbauer et al. (2015), Mezghani et al. (2018), Wilmink and Bockisch (2017), Dalpiaz et al. (2018), Niu et al. (2014), Femmer et al. (2017), Rosadini et al. (2017)
IV. Release Planning	del Sagrado et al. (2011), Li et al. (2010), Araújo et al. (2017), Baker et al. (2006), Mougouei and Powers (2020), Li et al. (2014), Zhang et al. (2007), Yang and Wang (2009), Przepiora et al. (2012), Mougouei (2016), Ngo-The and Ruhe (2008), Greer and Ruhe (2004), Pitangueira et al. (2016), Raatikainen et al. (2018), Pitangueira (2015), Pitangueira et al. (2017), Finkelstein et al. (2009)

Table 3.6.: Overview of the RE process and its tasks. The table includes references of relevant recommendation approaches used during the different activities of the process. Activities that are not an integral part of RE (such as the design, implementation, and testing of the software as well as the maintenance of the software project), are not shown in this table.

Requirements can be expressed in terms of textual descriptions, use cases, scenario descriptions, or mock-up illustrations of prototypical user interfaces. An old but still common way for requirements elicitation (Nuseibeh and Easterbrook, 2000) is that stakeholders have to write a structured requirements document and formulate their intentions and needs in the document using natural language. Once a requirements document is complete, the requirements engineers have to extract the requirements from the documents and to import the extracted requirements into a requirements management software tool (such as IBM DOORS). A very effortful challenge thereby is to distinguish between those paragraphs in a document which define a requirement and those which contain related content that has to be attached and linked to the requirements. In order to tackle this challenge, Falkner et al. (2019) present a recommendation approach which is based on supervised classification and assists requirements engineers in the automated extraction of requirements in large requirement document repositories. The approach analyzes text documents (Microsoft Word documents). It classifies paragraphs into requirements and descriptions. Paragraphs which cannot be classified with high confidence are marked as unclassified and have to be manually classified by a requirements manager. In addition to the aforementioned approach, Winkler and Vogelsang (2016) and Abualhaija et al. (2019) introduce comparable classification systems to identify requirements in requirement document repositories.

Another important task of requirements elicitation that poses a special demand for decision-support, is to group requirements into multiple categories (such as *functional vs. non-functional requirements* or *feature requests vs. software bugs*). There exist plenty of recommendation approaches that are based on supervised classification or data mining techniques that have been developed under the banner of RE research (Dalpiaz et al., 2019; Kurtanović and Maalej, 2017; Abad et al., 2017; Binkhonain and Zhao, 2019; Slankas and Williams, 2013; Mahmoud, 2015; Lu and Liang, 2017; Dhinakaran et al., 2018). These approaches use document classification techniques to classify requirements based on their textual description and content. For example, Casamayor et al. (2010) present a semi-supervised learning approach that focuses on the discovery of non-functional requirements in requirements specifications and exploits feedback from users as additional input source to enhance the overall classification performance. In recent years, more advanced document classification approaches have been introduced (Dekhlyar and Fong, 2017; Baker et al., 2019; Navarro-Almanza et al., 2017). These approaches use *deep learning*, *convolutional neural networks*, and pre-trained word embeddings to achieve higher levels of classification accuracy (Dekhlyar and Fong, 2017) compared to traditional classification approaches (such as *Naïve Bayes*, *Support Vector Machines*, and *Random Forest*).

The conventional way of requirements elicitation requires a direct communication between all stakeholders (in terms of interviews, workshops, questionnaires, or consultations with experts). However, the emerging trend of agile software development and web technologies has also influenced requirements elicitation practices. Nowadays, there exist additional information sources (e.g., social media channels or online forums) which can be exploited in order to extract and collect relevant requirements and related user feedback. In particular, online forums have a huge potential to drastically

improve requirements elicitation as these platforms allow to collect different opinions and viewpoints of a large number of stakeholders. For this reason, the inclusion of crowd-based user feedback in RE is known to be essential for the RE process (Groen et al., 2015; Pagano and Bruegge, 2013). This kind of feedback represents relevant technical information for developers (Williams and Mahmoud, 2017) that can help to identify missing requirements (e.g., new features, unknown bugs, etc.) and to improve the quality of the developed software. In contrast to the formal definition, the content of social media channels or online forums is often massive, fragmented, or represents inconsistent views of different stakeholders. For this reason, a variety of approaches has been developed that automatically extract requirements / features and important requirements-related information from social media channels (e.g., Twitter) (Williams and Mahmoud, 2017; Guzman et al., 2017), app stores, online product forums (Tizard et al., 2019), or online discussion forums (Khan et al., 2019, 2020; Kanchev et al., 2017a). These recommendation approaches help stakeholders to overcome the aforementioned challenges. For example, Khan et al. (2019) introduce a crowd-based RE approach called CROWDRE-ARG, which builds an argumentation model and identifies arguments for or against a certain requirement. Their model represents a useful recommendation tool to support requirements elicitation and the model is also capable of detecting conflicting arguments. Furthermore, the model helps to reason about the most influential arguments that triggered a certain requirement decision. The works of Kanchev et al. (2017b,a) and Khan et al. (2020) focus on the extraction of requirement-related information from online discussion forums. The presented approaches are potentially useful for requirements engineers and analysts to identify rationale elements, to enrich existing requirements with relevant annotations and to use these annotations in order to judge their decisions. Groen et al. (2015) propose an approach that combines existing requirements analysis and elicitation techniques. Their tool collects feedback created through social user collaboration and exploits data mining techniques in order to assist stakeholders in requirements elicitation. Oriol et al. (2018) present a tool called FRAME that follows a simultaneous collection approach of user feedback and usage data in mobile and web contexts. The collected data allows stakeholders to better understand the end-user needs and serves as a supportive means for continuous requirements elicitation.

As part of the European research project OPENREQ, Stanik et al. (2019) (see also Stanik and Maalej, 2019) developed techniques to identify requirements in social media channels using supervised classification. The work of Stanik et al. has been inspired by existing research in the field of extracting new features / requirements from app stores and online forums (Johann et al., 2017; Guzman et al., 2017; Williams and Mahmoud, 2017). The tool of Stanik et al. extracts requirements and requirements-related information from Twitter messages (called *tweets*) and app reviews¹¹. The underlying approach uses large labeled datasets (about 10,000 English and 15,000 Italian tweets from the telecommunication domain as well as about 6,000 app reviews) to train a recommendation model and classifies new messages into the classes *problem reports*, *inquiries*, and *irrelevant*. Thereby, the classes *problem reports* and *inquiries* represent requirements in terms of issues / bugs and feature re-

¹¹At the time of writing this chapter, the tool supports English and Italian.

quests, respectively. In contrast, messages classified as *irrelevant* are not considered as requirements. According to the evaluation results, the trained recommendation service achieved promising results on the aforementioned datasets using traditional machine learning as well as deep learning (in particular for English app reviews, *problem report*: precision: 0.83, recall: 0.75, f1-score: 0.79; *inquiry*: precision: 0.68, recall: 0.76, f1-score: 0.72; *irrelevant*: precision: 0.88, recall: 0.89, f1-score: 0.89). Recommendation approaches which are applied in the phase of requirements elicitation and definition often focus on recommending potentially relevant requirements from past projects that are similar to the current software project. The concept of re-using functionality between different software products has been inspired from the area of software product line engineering and the goal of most approaches is to foster the systematic reuse of requirements such that the efficiency of the RE process can be enhanced. Dumitru et al. (2011) introduce a content-based recommendation approach which extracts requirements from software repositories of past software projects and clusters them into requirement groups. These groups are then recommended to be reused in future software projects. The underlying algorithm extracts keywords from the current software project as well as past software projects and analyzes the overlapping ratio of the current project's keywords with other past projects. This is done to find the most similar projects from which yet unconsidered requirements can be recommended to be reused in the current project. By applying this approach, the overall costs of a software project (in terms of time, resources, and budget) as well as the risk of overseeing relevant requirements (i.e., completeness of the current project's requirements model) can be reduced. A more specific approach which is tailored to small-sized software businesses is explained by Ivan et al. (2016). The authors propose to use a software requirements catalog and present a reuse model for requirements which has been applied in the industrial context. Finally, Goldin and Berry (2015) introduce different reuse strategies of hardware and software requirement specifications which lead to a reduced time to market.

Requirements Negotiation and Analysis In the *requirements negotiation* phase, the requirements which should be implemented in the project are analyzed, discussed, and selected by stakeholders. According to Lim et al. (2010), a high degree of user involvement is crucial for the success of a software project. Consequently, an early involvement of stakeholders at the beginning of the RE process is essential. In order to ensure this, appropriate stakeholder experts, who should provide complete and well-defined descriptions to the existing requirements, should be assigned to the requirements (this can be regarded as being part of *requirements definition*). Moreover, another important task that falls into the category of *stakeholder identification / assignment* is the assignment of appropriate stakeholders to requirements, who should implement the requirements. This particular task of *stakeholder identification* can be regarded as being part of *requirements negotiation* as it often comes along with a collaborative evaluation and prioritization of the requirements. The assignment of appropriate stakeholders to each requirement is a very time consuming task for requirements engineers. Furthermore, the quality of the decision "*who to assign to which requirement*" mainly depends on these requirements experts. Hence, this task also requires much background and domain knowledge in order to be done correctly. In order to prioritize the requirements and to identify stakeholders who

should be in charge of implementing these requirements, a detailed evaluation and analysis of the requirements is indispensable. RS are appropriate and helpful tools to assist requirements engineers in such laborious tasks. The major objective to apply recommendation technologies in the context of *requirements negotiation and analysis*, is to improve the quality of stakeholder decisions and to offer automated support to evaluate / analyse requirements. This helps to reduce the risk of project delays and project cancellation.

Castro-Herrera and Cleland-Huang (2010) introduce a recommendation approach which recommends feature-requirements to stakeholders that match their individual interests and skills. Beyond that, the approach also tries to keep stakeholders informed about relevant issues. A completely different approach to recommend stakeholders for requirements is explained by Lim et al. (2010). The authors describe a recommendation approach called *StakeNet* which models recommendation relationships in node-edge-graphs that can be visualized as social networks. In such graphs, the stakeholders are represented as nodes and the directed edges express recommendations made by the stakeholders. This way, the graphs aim to answer the question "who recommended whom?". Thereafter, recommendations are generated based on the relations in the graph by exploiting different network analysis techniques (e.g., *betweenness centrality*) (see also Felfernig et al., 2013).

With an increasing number of requirements, the complexity of a group decision process becomes higher. Typically, software projects are often very complex and consist of several hundreds or thousands of requirements. In such cases, it is quite challenging and very time consuming to identify the most qualified stakeholders and software developers for every requirement. Another consequence of increased complexity are more decision conflicts between stakeholders caused by divergent opinions. Decision conflicts always indicate serious issues of some of the provided arguments which trigger discussions between the stakeholders that are involved in the conflict. The decision quality does not only depend on the different background and domain knowledge the respective decision-makers (stakeholders) have, but it also depends on many other factors, such as cognitive and personal biases (Maqsood et al., 2004; Tversky and Kahneman, 1975; Schulz-Hardt et al., 2006). In general, such bias effects become more serious whenever the complexity of a decision task and / or the time pressure for decision-making is high. Group recommendation approaches are capable to be applied in this context to mitigate such bias effects (Tversky and Kahneman, 1975; Schulz-Hardt et al., 2006; Samer et al., 2020) and to support a collaborative analysis, evaluation, and prioritization of requirements. Furthermore, group RS can be effectively applied to resolve decision conflicts between stakeholders. For example, Felfernig et al. (2011) analyzed the impact of the application of such group recommendation technologies on the outcome of this group decision process and show that these systems are well-suited to improve the quality of decision support. The prioritization of requirements is a crucial task as the amount of available resources in software projects are typically limited. There exists a large variety of requirements prioritization approaches (Tonella et al., 2013; Karlsson, 1996; Ruhe et al., 2002; Duan et al., 2009; Perini et al., 2013; Sadiq et al., 2017) based on different concepts (such as *genetic programming* and *negotiation-based solutions*) and algorithms (such as *analytic hierarchy*

process and case-based ranking).¹²

Ninaus et al. (2014) present a group recommender system that supports group decision making in the context of requirements prioritization. The authors' work is based on the concept described in Felfernig and Ninaus (2012). In their evaluation approach, stakeholders evaluate and analyse requirements based on interest dimensions such as *associated risk, feasibility, and costs* (see also Section 3.5.2). Moreover, Samer et al. (2020) developed a group recommendation approach which focuses on improving requirements prioritization by making the preference elicitation process of stakeholders more flexible. The authors introduce user interfaces that foster information exchange among stakeholders and use liquid democracy (delegate voting) (Johann and Maalej, 2015; Zhang and Zhou, 2017; Atas et al., 2018) to allow stakeholders to transfer voting tasks to experts based on the level of interest dimensions. Moreover, the group RS of Samer et al. also features mechanisms (i) to resolve decision conflicts between stakeholders and (ii) to hide / obscure preferences of the individual group members at the beginning of the evaluation process in order to counteract cognitive biases such as anchoring effects. The group recommendation approach was evaluated in a large-scale user study (N=313 participants) and the evaluation results indicate that argumentation-based user interfaces lead to an increased interaction and communication activity in stakeholder groups (around 2.3 times higher on average compared to a standard one-dimensional 5-star rating scheme). Apart from an increased information exchange among stakeholders and more user interactions, a statistical analysis of the study results also reveals that stakeholders tend to adapt and update their (textual) arguments more often which effectively supports the resolution of decision conflicts. The bottom line of the study is that argumentation-based group recommendation approaches can pave the way for high-quality requirement evaluations to support a collaborative evaluation, negotiation, and prioritization of software requirements.

Some further research conducted within the scope of the OPENREQ project, focuses on the recommendation of relevant requirements to volunteer software developers in open-source projects. Felfernig et al. (2018) introduce a content-based recommendation approach which recommends suitable requirements (in terms of issues / bugs) to the stakeholders / developers. The recommended requirements consider relevance for the community (i.e., priority) as well as suitability for the developer (i.e., *"Is the topic of the requirement of interest for the developer?"*). Their approach involves *content-based filtering* as well as recommendation aspects stemming from *multi-attribute utility theory* (Wallenius et al., 2008). A known issue / drawback of this approach is that it requires a sufficient amount of user data in terms of requirements which have been resolved by the developer in the past. With insufficient developer-related data, it is not possible for the approach to create accurate user profiles. This problem is a common issue of RS and referred to as the *cold-start problem* (see also Section 3.7). An approach which tackles the cold-start problem in this specific scenario has been presented by Stanik et al. (2018). The authors describe this scenario as *onboarding* and show appropriate solutions to recommend requirements (issues / bugs) to newcomers (i.e., new developers). Their approach is based on *supervised classification* and *natural language processing* (NLP) and can be used in issue

¹²For a more detailed overview, we refer to Riegel and Doerr, 2015 and Qaddoura et al., 2017.

tracking environments (such as *Bugzilla* or *Jira*). In the same line of research, Samer et al. (2019) also introduce a classification-based recommendation approach for issue tracking. While the approach developed by Stanik et al. (2018) generates recommendations which specifically target the group of newcomers to support onboarding as well as user retention, the approach of Samer et al. (2019) produces user-specific recommendations for all kinds of developers. The approach of Stanik et al. (2018) has been evaluated on three large datasets of open-source projects (QT (N=55610), ECLIPSE (N=158843), and LIBREOFFICE (N=10958)) using different classification algorithms (*Random Forest*, *Naïve Bayes*, *Decision Tree*, *Support Vector Machines*) with the aim of generating *non-personalized* recommendations of requirements / issues suitable for any type of newcomer. The results of the first evaluation part which focused on newcomers' onboarding revealed that *Random Forest* achieved the highest prediction results on all datasets, in terms of precision (up to 0.91 precision, f1-score: 0.72). The focus of the second evaluation part was on newcomers' retention and according to the results, *Random Forest* and *Decision Tree* performed the best in terms of precision, recall, and f1-score (precision and recall of up to 0.92, f1-score: 0.91). In contrast to the approach presented by Stanik et al. which generates *non-personalized* recommendations for any type of newcomers, the RS developed by Samer et al. (2019) produces *personalized* recommendations that best match the preferences of a developer. The RS is a prioritization tool for the ECLIPSE community that supports the issue management platform BUGZILLA and provides decision-support for open-source developers to find the next best matching requirements / bugs / features. The authors' approach has been evaluated on three large datasets of different open-source projects (ECLIPSE (N=141117), MOZILLA (N=751961), and LIBREOFFICE (N=47542)) and the evaluation results indicate that the approach performs significantly better with *Random Forest* classification (in case of ECLIPSE, precision: 0.88, recall: 0.55, f1-score: 0.68; in case of MOZILLA, precision: 0.73, recall: 0.49, f1-score: 0.59; in case of LIBREOFFICE, precision: 0.81, recall: 0.75, f1-score: 0.78) than other comparable approaches and baselines (see Samer et al., 2019). The RS comes along with a plugin for the ECLIPSE development environment. The ECLIPSE plugin presents a user interface that allows its users to directly interact with the RS in order to provide feedback to the recommendations proposed by the system (for more details, we refer to Section 3.6.2).

In the context of developing ultra-large-scale software systems with a very high number of requirements and stakeholders involved, Castro-Herrera et al. (2008) as well as Mobasher and Cleland-Huang (2011) provide solutions based on clustering to group requirements and then to recommend stakeholders to the clusters by using content-based recommendation. Their clustering-based recommendation approach ensures a decent stakeholder coverage. Kifetew et al. (2017) describe a completely different collaborative requirements prioritization approach which is based on the idea of gamification. The presented tool uses game elements and genetic algorithms to encourage stakeholders to contribute to the group decision-making process of requirements prioritization. Another recommendation approach which can be applied in the context of general large-scale industrial projects is introduced by Samer et al. (2018). The approach exploits content-based recommendation techniques to improve the col-

laborative prioritization of requirements among stakeholders. The described stakeholder assignment scenario allows *human stakeholders* (who are candidates proposed by the requirements manager) to propose (further) candidates for a requirement and to evaluate all proposed candidates. In addition, *artificial stakeholders* (represented by different recommendation systems) are used to automatically propose further candidates and to evaluate all proposed candidates. The proposed stakeholders are then evaluated with respect to the dimensions *appropriateness* and *availability* by themselves, the requirements managers, and the *artificial stakeholders*. The dimension *appropriateness* indicates the suitability of the candidate for the requirement, whereas the dimension *availability* signifies whether the candidate has enough time and resources to take over responsibility for the requirement. Furthermore, the proposed candidates can accept themselves as final candidates, raise a veto, and / or propose further candidates. Finally, the proposed candidates with the highest overall evaluation scores are then suggested to the requirements manager to be assigned to the requirement. This way, the whole stakeholder assignment process can be improved and accelerated such that the overall effort for all involved stakeholders as well as the chance of overseeing suitable stakeholders can be reduced for the requirements managers.

Quality Assurance The *quality assurance* phase deals with the aspect-oriented evaluation of a project's requirements. In this context, aspects which should be evaluated, represent qualitative attributes, such as the *feasibility* (technical vs. economic feasibility), the *completeness* (all relevant requirements are included in the requirements model), the *consistency* (no requirements conflict with each other), the *understandability* (readability / understandability quality of the requirement's description), and the *reusability* (for future software projects) (Felfernig et al., 2013).

An example of a RS which addresses the *consistency* aspect of a requirements model are *inconsistency recommenders* (Iyer and Richards, 2004; Felfernig et al., 2010). Their goal is to detect inconsistencies which can exist between some requirements of a software project and to recommend these as issues that need to be fixed. In practice, inconsistencies usually occur if stakeholders do not have enough time to check the consistency of the requirements model, their expertise and knowledge is limited to certain topics, or their perceptions diverge from each other (Iyer and Richards, 2004). Felfernig et al. (2010) describe a knowledge-based recommendation approach which automatically detects inconsistencies and provides solutions on how to resolve inconsistencies. These solutions are presented in the form of diagnoses which can be interpreted as instructions on how to repair the existing requirements model. Such diagnoses are minimal sets of requirements or stakeholder preferences which need to be adapted in order to resolve an inconsistency. In order to offer precise repairs, such an inconsistency recommendation solution requires a complete requirements model, a complete definition of all model constraints, and all individual requirement preferences of the stakeholders must be known (i.e., priorities). Felfernig et al. (2013) introduce a collaborative recommendation approach, which recommends relevant constraints. The underlying assumption is that if stakeholders try to understand a set of constraints related to a requirements model, a RS can recommend related constraints using collaborative filtering. Such recommendations can be constraints that have been investigated by other stakeholders

with a similar constraint browsing behavior. In this context, constraints which have been analyzed by a stakeholder are associated with positive ratings (i.e., rating = 1). Beyond the scope of RS, Parra et al. (2018) introduce a visualization approach that measures the evolution of the requirements quality. The approach uses quality metrics to assess requirements correctness, completeness, and consistency. The determined quality information is visualized in charts and represents a basis for providing suggestions to stakeholders on how to improve the quality of the requirements model.

Apart from the aforementioned model-related inconsistencies, requirements defined in natural language are also prone to suffer from textual inconsistencies. Such inconsistencies often occur in terms of textual ambiguities that can also pose a threat to the understandability of a requirement. While a long stream of RE research presents frameworks and guidelines on how to analyze and detect ambiguities (Gupta and Deraman, 2019; Gervasi et al., 2019; Sabriye and Zainon, 2017; Ferrari et al., 2016; Massey et al., 2014; Bano, 2015), some work also introduces tools that support stakeholders in finding terminological ambiguities in requirements (Ferrari and Gnesi, 2012; Rosadini et al., 2017; Wilmink and Bockisch, 2017; Ferrari et al., 2018). For example, Gleich et al. (2010) provide a tool that detects ambiguities and explains corresponding sources of the detected ambiguities. Dalpiaz et al. (2018) contribute an NLP-based approach that identifies textual defects and ambiguities in existing requirements. Moreover, their approach also addresses the incompleteness aspect by detecting missing requirements. Another NLP-based ambiguity detector that uses the concept of word embeddings has been developed by Ferrari and Esuli (2019). The authors' approach tries to find ambiguous words between different stakeholder domains and rank these words by a calculated ambiguity score.

Beyond the scope of ambiguity detection, further approaches and techniques can be applied to improve the quality of requirements. Femmer et al. (2017) describe a basic requirements validation approach to detect quality violations in textual descriptions of requirements. The authors' approach performs an automatic analysis (using, e.g., part-of-speech tagging, morphological analysis, and dictionaries) in order to find and report relevant defects ("requirement smells") in requirements. Tiwari and Laddha (2017) developed a light-weight validation tool that focuses on use case based textual specifications of functional requirements. The tool automatically assesses and validates the quality of use cases, highlights errors, checks for completeness regarding the use cases, and provides suggestions to fix quality issues / smells. Danylenko and Löwe (2012) present a general approach to foster the development of more qualitative software applications. Their work primarily addresses the design of efficient and fast software systems (e.g., computational software), and focuses on non-functional requirements which relate to the efficiency of the system. The authors developed a context-aware RS that aims to improve the software development process of such systems by helping stakeholders (in particular, system designers and developers) to reduce the overall development effort in terms of time. Cleland-Huang et al. (2009) describe a similarity-based approach which clusters similar requirements in order to detect redundancies in the existing requirements model and to support the prioritization of feature requests. In particular, the approach of Cleland-Huang et al. represents a RS to support quality assurance in the context of large feature sets. Moreover, Fitzgerald et al. (2011) exploit ma-

chine learning techniques in order to reduce software failures and, hence, to improve the quality of a software project. A more recent recommendation approach for industrial applications has been introduced and developed by Mezghani et al. (2018). The presented approach is based on clustering and uses the k-means algorithm to detect and highlight redundancies and inconsistencies between requirements. The recommendations that lead to a reduction of inconsistent and redundant requirements help requirements engineers to pave the way for better quality assurance. Another important issue of RE quality assurance represents the traceability of requirements throughout the entire software development cycle (i.e., life of a requirement backward and forward). In particular, requirements traceability is relevant for the development of safety-critical systems where standards and guidelines (such as ISO 26262 or Automotive SPICE) require the recording of requirements traceability in order to be able to prove that critical requirements have been implemented and validated in an appropriate manner. Since requirements traceability is very time-intensive for humans, automated solutions are required to assist stakeholders in this task. Mezghani et al. (2019) present a clustering-based approach that clusters and suggests linked requirements to support stakeholders in traceability management. Niu et al. (2014) propose a recommendation approach to support traceability-enabled software refactorings. Their approach identifies places inside software projects that should be refactored and determines what refactorings are appropriate to be applied for the corresponding identified places. Moreover, further approaches related to requirements traceability aim at reaching various goals, e.g., to support an automated trace recovery or to support software product line development by building software product portfolios consisting of similar product variants (Cleland-Huang et al., 2007; Mahmoud and Niu, 2010, 2011; Keenan et al., 2012; Linsbauer et al., 2013, 2015).

The complete and gapless definition of dependencies between requirements is essential in order to obtain consistent requirements models. The frequent occurrence of dependencies between requirements has been demonstrated by Vogelsang and Fuhrmann (2013). The result of their work also shows that the identification of dependencies is a challenging and time-consuming task. The authors mention that most stakeholders are not aware of a large number of dependencies that can exist between requirements. Moreover, requirement dependencies are more the rule than an exception in real-world projects (Carlshamre et al., 2001; Vogelsang and Fuhrmann, 2013). Hence, an in-depth analysis of the relevant requirements is essential to find all correct dependencies between the requirements. The main challenges of detecting dependencies between requirements are two-fold, being (i) detecting and (ii) handling these in subsequent release planning activities. The latter will be addressed later in this section (see *release planning*). The former is dedicated to the currently discussed phase of the RE process, where stakeholders have to manually find and define (i.e., manually detect) correct dependencies carefully and as early as possible. It is important to point out that in the case of incorrect, incomplete, or inconsistent dependencies, a software project can most probably not be completed successfully. Most existing and past work in this research area follows the idea of detecting similarities between requirements (Ninaus et al., 2014; Felfernig et al., 2013; Natt och Dag et al., 2002) using basic content-based recommendation, information retrieval, fuzzy logic, or clustering techniques.

A more optimized and advanced content-based dependency recommendation approach which goes beyond these basic concepts, is presented by Atas et al. (2018). In their paper, the authors introduce a tool based on supervised classification and natural language processing (NLP) that identifies dependencies between requirements. The tool examines all possible pairs of requirements in a project and then classifies them into positive and negative dependency candidates whereby the positive candidates represent those pairs where the tool thinks that there exists a real dependency. In general, there are different types of dependencies such as *requires*, *excludes*, *part-of*, etc. (see Atas et al., 2018). The tool of Atas et al. focuses on the dependency type *requires* which appears as the most frequent dependency type in software projects (Ferber et al., 2002; Atas et al., 2018; Deshpande et al., 2019). In this context, a *requires*-dependency R_x *requires* R_y (denoted as $R_x \rightarrow R_y$) describes the relationship that the requirement R_x can not be implemented before R_y has been implemented. Furthermore, dependencies of type *requires* are unidirectional (i.e., the expression $R_x \rightarrow R_y$ is not equivalent to the expression $R_y \rightarrow R_x$). It is important to point out that the aforementioned tool is not only able to detect the existence of a dependency but also the correct direction of the dependency. Deshpande et al. (2019) present another recommendation approach that also automatically extracts requirement dependencies. An initial evaluation of the approach using three classification algorithms (*Random Forest*, *Naïve Bayes*, and *Support Vector Machine*) showed that all three algorithms achieved similarly high accuracy levels (f1-scores of all three classifiers were around 0.7). By applying *weakly supervised learning* (WSL) (Zhou, 2017) to generate pseudo labels for the unlabeled data samples, the performance of the trained models could be further improved (f1-scores around 0.85). Samer et al. (2019) introduce two refined approaches to the automated detection of requirement interdependencies. They have developed two approaches – one approach feeds a supervised classification model with labeled dependency samples and the other one is based on unsupervised learning that can be directly used with raw unlabeled requirement samples. In contrast to the work presented by Atas et al. (2018) and Deshpande et al. (2019), the supervised classification approach developed by Samer et al. (2019) utilizes new feature types and statistical concepts from the area of *information theory*. Moreover, the evaluation yields improved and more reliable prediction results of the identified dependencies on the same dataset compared with the solution of Atas et al. (2018). The unsupervised learning approach of Samer et al. is based on *Latent Semantic Analysis* (Deerwester et al., 1990) and represents a soft-clustering technique that groups related requirements based on underlying content-related concepts. It uses the (textual) content of the requirements and transforms the descriptions into a low dimensional space representation. Each dimension of the reduced space representation corresponds to a content-related concept. The reduction of dimensions helps to filter out noise and to group semantically-related requirements. The approach determines requirement pairs which are closely related to each other in the semantic space and considers such pairs as dependent pairs (i.e., both requirements of a pair are dependent). Similar to the WSL-based approach of Deshpande et al. (2019), this unsupervised dependency detection approach can be used with unlabeled samples and it can be applied once the stakeholders have defined all requirements in a software project. Consequently, this represents a simple and powerful

approach to reduce costly efforts regarding the querying of human experts in order to label a large set of requirements.

Release Planning Prioritized requirements can be assigned to releases. Due to the limited amount of available resources, *requirements triage* needs to be applied in the context of release planning (Davis, 2003). The concept of *triage* stems from the medical domain where patients are usually categorized into three types: patients who will die, patients who will survive, and patients who can survive when appropriate medicine is given to them. In the context of release planning, the approach is similar. There exist requirements which must not be assigned to the next release, requirements which must be included in the release, and requirements whose assignment to the next release is optional. In this context, the selection of a subset of suitable requirements to be implemented in the next release is called the *next release problem* (NRP) (Bagnall et al., 2001). The suitability of a requirement for a release depends on pre-defined constraints that, for example, aim to maximize the total revenue by also keeping the costs within the limits of the budget available for the release. These requirements have to meet specified criteria which include, for example, the expected revenue of the release or constraints such as budget limitations.

Regarding recommendation tools which support release planning, there exist numerous research contributions (Zhang et al., 2007; Yang and Wang, 2009; Finkelstein et al., 2009; Li et al., 2010; del Sagrado et al., 2011; Pitangueira et al., 2016). For example, Raatikainen et al. (2018) introduce a recommendation approach based on a knowledge-based configuration. In order to compute a set of possible / feasible solutions for release plans, their prototype considers all defined constraints of the requirements model and the dependencies which exist between the requirements. The computed release plans are recommended to the requirements engineers. In case of inconsistencies (e.g., conflicting dependencies between the requirements), repairs (in terms of diagnoses) to resolve the inconsistencies are recommended. Przepiora et al. (2012) propose a basic hybrid approach that combines a knowledge-based approach (also supporting requirement dependencies) with a release planning tool called RELEASEPLANNER. Baker et al. (2006) apply two *search-based software engineering* approaches (Harman et al., 2012) using greedy and simulated annealing algorithms to tackle the NRP. The authors also demonstrate that both approaches have the potential to outperform the ranking of human experts. Moreover, Mougouei and Powers (2020) present an approach to support stakeholders in release planning. Since the value / benefit of an individual software feature / requirement for an end-user depends on the users' preferences, it is important to determine a set of most relevant features that can be assigned to the next release. The approach of Mougouei and Powers aims to identify such value-related dependencies by mining user preferences for software features. Furthermore, the approach uses dependencies in order to determine an optimal subset of software features to be assigned to a release of a software product. In an earlier work of Mougouei (2016), the author proposes an approach for requirements selection that considers requirement dependencies and the strength-level of these dependencies determined by using a graph-based dependency modeling approach.

Greer and Ruhe (2004) introduce an approach called EVOLVE that supports incremental software de-

velopment processes. Their approach utilizes an iterative optimization technique that is based on a genetic algorithm and assists stakeholders in the continuous planning of software projects by recommending a set of qualified candidate solutions (i.e., release plans). Ngo-The and Ruhe (2008) introduce a more general approach to solve planning problems (also apart from release planning, e.g., reuse of components or selection of architectures). The authors devised a recommendation tool called EVOLVE+ which extends EVOLVE. In contrast to EVOLVE+, EVOLVE adds diversification to mitigate the uncertainties of the planning problem formulation. In other words, diversification is used to improve (release planning) decisions and the tool also provides guidance when selecting a release plan. Pitangueira et al. (2017) describe a multi-objective approach to tackle the NRP. The approach of Pitangueira et al. aims at improving the decision quality by minimizing the risk of stakeholder dissatisfaction. Another approach based on multi-objective optimization is presented by Li et al. (2014). In contrast to traditional multi-objective optimization, the approach exploits the Monte-Carlo Simulation technique to optimize requirement choices considering three different aspects which are cost, revenue, and uncertainty. In order to find an optimal subset of candidate requirements that satisfies the users' demands, Pitangueira (2015) proposes an interactive approach to incorporate the preferences of different stakeholders that assists users in the requirements selection process. Moreover, Araújo et al. (2017) present a *search-based software engineering* approach which uses an interactive genetic algorithm. In order to be more effective by taking the preferences and experiences of the decision makers into special consideration, the approach utilizes a learning mechanism to reduce the number of user interactions and enhances the optimization process.

3.6.2. Requirements Management Platforms

There exist many well-known and broadly used requirements management tools, such as MODERN REQUIREMENTS¹³, IBM DOORS¹⁴, JAMA SOFTWARE¹⁵, ATlassian JIRA¹⁶, etc. These tools provide techniques to facilitate RE tasks as well as the maintenance of requirements defined by stakeholders. Due to a continuously increasing number of requirements management tools and a diverging variety of supported tasks (including, e.g., requirements elicitation, requirements traceability, requirements prioritization, requirements testing, stakeholder reviewing), the selection of the right RE management tools is often very challenging. Hoffmann et al. (2004) and Alghazzawi et al. (2014) propose a list of criteria to select appropriate RE management tools according to the tool capabilities and the conceptions that meet the individual needs of a project team. Apart from the differences in functionalities provided by different RE tools, many do not exploit the full potential of integrating recommendation technologies. The remainder of this section briefly introduces a selected portion of two requirements management platforms. In addition, we present requirements management tools that support common RE tasks relevant for open-source communities. INTELLIREQ (Ninaus et al.,

¹³MODERN REQUIREMENTS: <https://www.modernrequirements.com>

¹⁴IBM DOORS: <https://www.ibm.com/products/requirements-management>

¹⁵JAMA SOFTWARE: <https://www.jamasoftware.com>

¹⁶ATLASSIAN JIRA: <https://www.atlassian.com/en/software/jira>

2014) and OPENREQ!LIVE (Samer et al., 2020) are freely available and provide an easy-to-use web user interface which facilitates stakeholder interactions with the provided recommendation tools.

IntelliReq

The INTELLIREQ environment is based on different recommendation approaches that support stakeholders in requirements-related tasks, such as *requirements definition*, *quality assurance*, *requirements reuse*, and *release planning*. The web user interface of INTELLIREQ allows (technical and also non-technical) stakeholders to access different RE tools that deliver the recommendation power to improve decision-making in RE-related tasks. For example, INTELLIREQ comes along with a group-based evaluation and recommendation system for requirements prioritization which is discussed by Ninaus et al. (2014). For each requirement, the group RS proposes recommendations to help a group of stakeholders to jointly decide on the final prioritization of a requirements list (see Figure 3.2). The system also provides functionality to detect and show evaluation conflicts which indicate stakeholder disagreement regarding a specific interest dimension. A conflict occurs whenever two or more stakeholders evaluate an interest dimension differently. For each interest dimension, the system shows a traffic light symbol which appears *red* if there exists a conflict for the respective interest dimension and *blue* if there are no conflicts. Such a conflict can be, for example, that a requirement is rated to be infeasible by stakeholder A but considered as completely feasible by stakeholder B. By using the traffic light symbols, the stakeholders involved in the conflict are now informed about the conflict and are encouraged to negotiate and reevaluate the requirement with respect to this interest dimension. Major advantages caused by the application of INTELLIREQ's recommendation technologies in RE are (1) an increased reuse of requirements, (2) active guidance of stakeholders, (3) increased consistency in requirements models, and (4) reduced time efforts needed for the construction of requirements models (Ninaus et al., 2014). For an in-depth discussion of recommendation approaches applied in INTELLIREQ as well as the results of empirical studies that show in which way recommenders can improve the quality of RE processes, we refer the reader to Ninaus et al. (2014).

OPENREQ!LIVE

OPENREQ!LIVE is a free web-based platform which offers direct access to a recommendation toolchain that has been developed within the scope of the Horizon 2020 research project OPENREQ¹⁷ (Palomares et al., 2018; Felfernig et al., 2017). While INTELLIREQ only supplies a reduced set of basic recommendation functionalities for specific RE tasks where the applied recommendation approaches do not go beyond the level of semi-automated learning, OPENREQ!LIVE exploits recommendation techniques that support a large variety of RE tasks that cover the RE process as a whole. The OPENREQ toolchain includes a broad collection of different recommendation functionalities. These functionalities are based on recommendation approaches which have been discussed

¹⁷OPENREQ: <https://openreq.eu>

Meta Data

Description - Bluetooth

The watch needs a bluetooth connection. This is necessary to connect the watch to other devices. Example: heart rate chest strap

Meta Data	Adjust	Info
Risk: 4 (Average)	<input type="range"/>	
Feasibility: 8 (High)	<input type="range"/>	
Cost: 5 (Average)	<input type="range"/>	
Relevance: 4 (Average)	<input type="range"/>	
Priority: 1 (Low)	<input type="range"/>	
Duration: h	<input type="text" value="20"/>	

Preferred Release: R2 - 2014-03-13

Select Meta Data Type to view

Priority

All ratings for Priority

Picture	Name	Rating
	IRManagement	4
	IRKunde	7
	AlexanderFelfernig	1

Recommendation

Majority Recommendation: 1

Average Recommendation: 4

Save

Figure 3.2.: Example of a group-based requirements evaluation scenario in INTELLIREQ (see Ninaus et al., 2014). The traffic light feedback mechanism indicates an inconsistency with respect to the three stakeholder ratings provided for the property *priority* (red light).

in Section 3.5. The most relevant functionalities integrated into OPENREQ!LIVE, have already been presented in Section 3.6.1. In order to provide stakeholders a convenient and user-friendly access to the OPENREQ's recommendation features, the web-based RE platform OPENREQ!LIVE has been developed. OPENREQ!LIVE is a collaborative RE platform which proactively supports the cross-fertilization of different ideas shared between stakeholders. The platform allows stakeholders to manage and maintain their projects and take advantage of the recommendation features provided by OPENREQ's services. OPENREQ!LIVE assists stakeholders in a variety of common everyday RE tasks stakeholders have to face in their software projects. The platform comes along with functionality to create and maintain a requirements model of a software project. In OPENREQ!LIVE a requirements model includes requirements, dependencies (between requirements), software releases, and release-

specific constraints (e.g., a strict release deadline or the maximum capacity of requirements a release can handle) of a project. In OPENREQ!LIVE, stakeholders can directly update and modify the requirements and releases on a single project page. Moreover, the system allows stakeholders to search for and filter certain types of requirements. This allows stakeholders to keep track of recent and most relevant changes in a requirements model. Figure 3.3 shows the main page of an example software project. The page provides a compact overview of the project which presents the project structure defined by the stakeholders. Requirements consisting of a unique ID, title, description, and status, are listed on the page and ordered by a utility value. Each release has a deadline and a maximum capacity value (in hours) which limits the possible number of requirements that can be assigned to the release. The red labeled numbers indicate issues (e.g., requirement duplicates and ambiguities in a requirement’s description text) reported by some of the OPENREQ services.

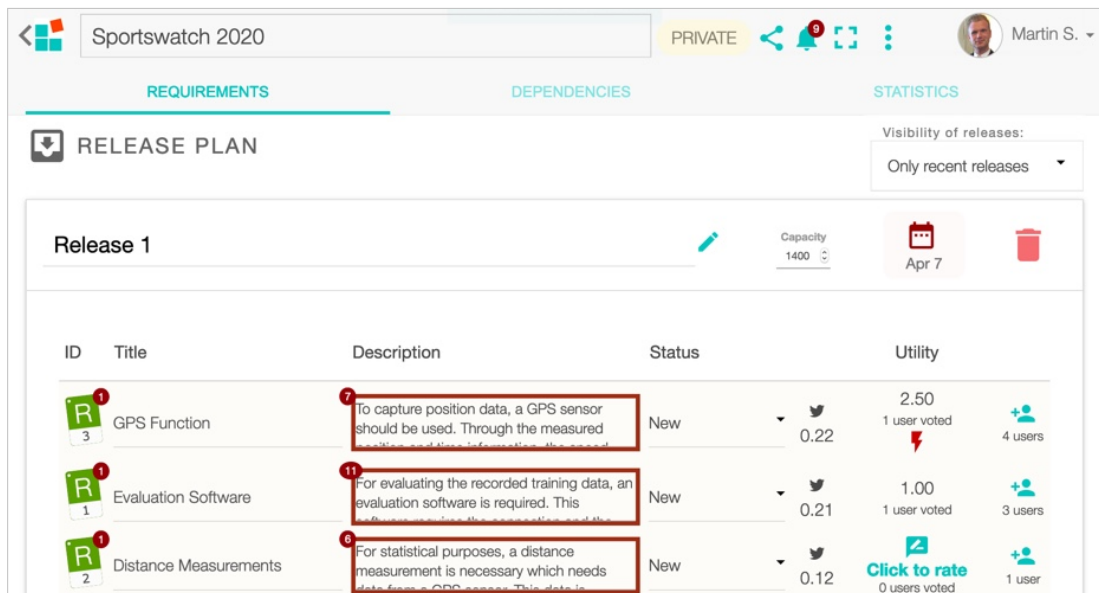


Figure 3.3.: OPENREQ!LIVE project overview.

Major tasks that can be supported via OPENREQ!LIVE are:

1. Requirements Elicitation

In OPENREQ!LIVE, projects can be connected to social media channels (at the moment, OPENREQ!LIVE supports TWITTER). In the background, the channel identifiers are automatically passed to a recommendation service developed by Stanik et al. (2019) (see also Stanik and Maalej, 2019) and description in Section 3.6.1). The recommendation service uses supervised classification to identify and classify relevant tweets in the social media channels. Relevant tweets identified as requirements are then recommended to the stakeholders.

2. Prioritization of Requirements

Figure 3.4 shows an example of an argumentation-based rating interface. Stakeholders are asked to provide textual feedback (in terms of arguments) for every requirement. After the stakeholders have evaluated all requirements, the group recommendation system generates a ranking of prioritized requirements that follows the approach of Samer et al. (2020) described in Section 3.6.1.

3. Stakeholder Recommendation

Automatic recommendations for appropriate stakeholders are marked with an "AI" tag whereas manually assigned stakeholders are marked with an "ST" tag (see Figure 3.5). In order to determine final stakeholder candidates for a requirement, the applied group recommendation approach involves human stakeholders as well as artificial stakeholders in the evaluation process. The underlying algorithm is based on the concept presented by Samer et al. (2018) in Section 3.6.1.

4. Full-fledged Dependency Management

Dependencies can be added in different ways in OPENREQ!LIVE. First, dependencies can simply be added manually by defining the left and the right side of the dependency as well as the dependency type (i.e., *implies*, *requires*, *excludes*, or *incompatible*). Second, dependencies can be imported by OPENREQ recommendation services (see *Quality Assurance* in Section 3.6.1). In case of inconsistent dependency definitions, the system informs the user about the current state. OPENREQ!LIVE presents the details as well as the causes of the inconsistencies. In addition, OPENREQ!LIVE indicates ways to restore consistency (adaptation of the requirements and / or adaptation of the dependencies) – see Figure 3.6.

5. Advanced Statistics

In a project's statistics section of the OPENREQ!LIVE user interface, details about the change history of a requirement are shown. Furthermore, the user sees a visualization of all dependencies which shows the relation to the requirements in a very intuitive fashion.

Apart from these core tasks, OPENREQ!LIVE also features further tools which support other RE-related tasks, such as *ambiguity detection* (quality assurance), *requirement similarity detection* (e.g., to identify related requirements and duplicates), and *release planning*. For a more detailed overview of the recommendation services integrated into OPENREQ!LIVE, we refer to Samer et al. (2020).

Requirements Management Platforms for Open-Source Communities

The coordination and planning of open-source projects as well as of commercial software projects in an agile development process always requires a way to communicate goals, plans, and issues between project managers, product owners, developers, and users. These jobs can be realized by using issue

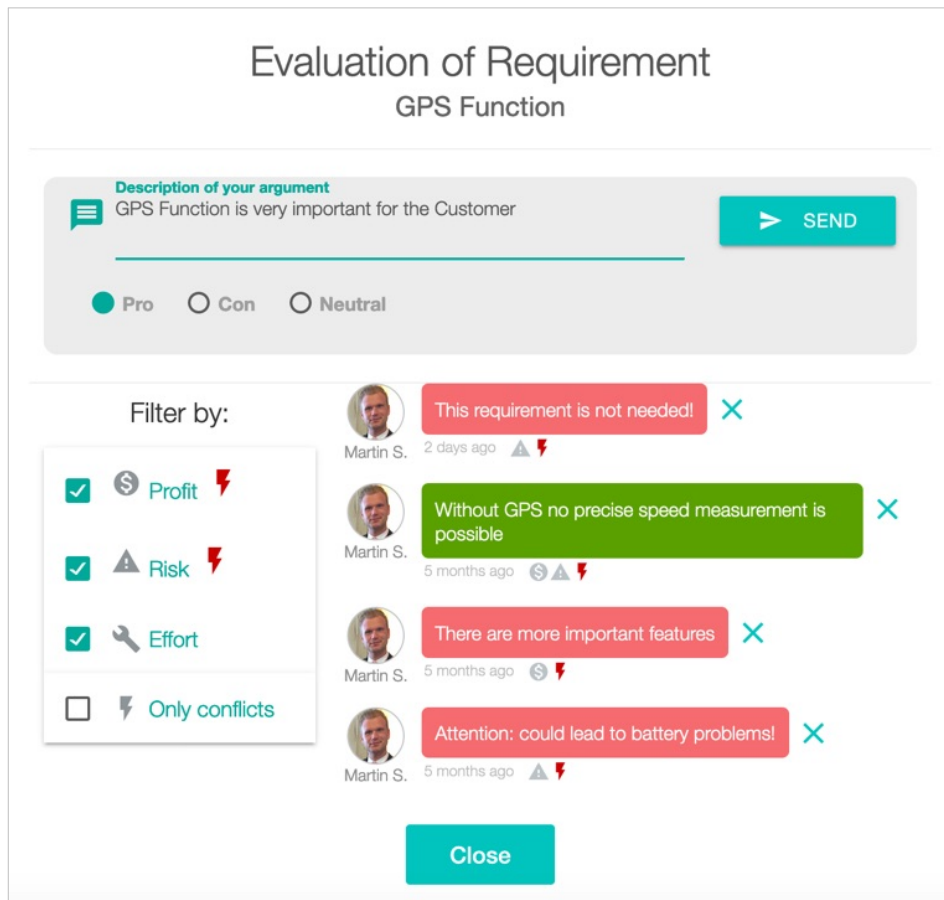


Figure 3.4.: Argumentation-based rating interface. Each argument must be assigned to at least one interest dimension. Negative arguments are highlighted in red, positive ones in green, and neutral arguments in orange.

tracking systems such as, for example, JIRA, BUGZILLA, or GITHUB. The requirements (represented as *issues* in such systems) are used as a basis for decision-making.





Recommenders in QT Company The QT company (a project partner of the OPENREQ project) uses Jira to collect and manage requirements, features, and bugs for both their open-source and commercial software projects. Due to the fact that every day many more issues are being added to such systems, keeping track of all existing issues in issue tracking systems is very cost-intensive. In most cases, these issues are manually checked, triaged, and maintained. One big challenge is adding dependencies (often called links) in issue tracking systems because this assumes that the maintainer knows all relevant issues.

Within the scope of OPENREQ (Palomares et al., 2018; Felfernig et al., 2017), a JIRA plugin has been developed (Lüders et al., 2019) that supports all stakeholders (product / project managers, developers, and users) in identifying dependencies and detecting inconsistencies between requirements. In

Propose assignee:

Enter name... + PROPOSE

Proposed assignees:

	✖ Appropriateness		🕒 Availability		Result	
	Your	Average	Your	Average		
 Martin Stettinger (you) AI Accepted	9	3.6	4	7.0	5.3	
 Muesluem Atas AI	10	3.3	1	7.8	5.5	
 A F ST	2	2.0	9	9.0	5.5	✕
 Lord Root	-	2.8	-	8.3	5.5	

Close

Figure 3.5.: Stakeholder Assignment. In case a stakeholder accepts the assignment he / she will be marked in green.

Requirement Model Inconsistencies

Conflicting Dependencies:

[R3] GPS Function

To capture position data, a GPS sensor should be used. Through the measured position and time information, the speed and the distance can be measured.

≥

[R1] Evaluation Software

For evaluating the recorded training data, an evaluation software is required. This software requires the connection and the access to the clock's internal memory. The evaluation should contain measured information regarding the distance, the height, the average heart rate, and the calorie consumption.

How to get consistent?

Adapt Requirements
Adapt Dependencies

The following dependencies have to be removed:

[R3] GPS Function

To capture position data, a GPS sensor should be used. Through the measured position and time information, the speed and the distance can be measured.

≥

[R1] Evaluation Software

For evaluating the recorded training data, an evaluation software is required. This software requires the connection and the access to the clock's internal memory. The evaluation should contain measured information regarding the distance, the height, the average heart rate, and the calorie consumption.

Figure 3.6.: Details about the inconsistencies of the current requirements model.

addition, a novel visualization technique has been implemented that presents the requirements with the corresponding dependencies. The plugin visualizes the links between issues, recommends links between issues and checks the consistency of release planning (see Figure 3.7).

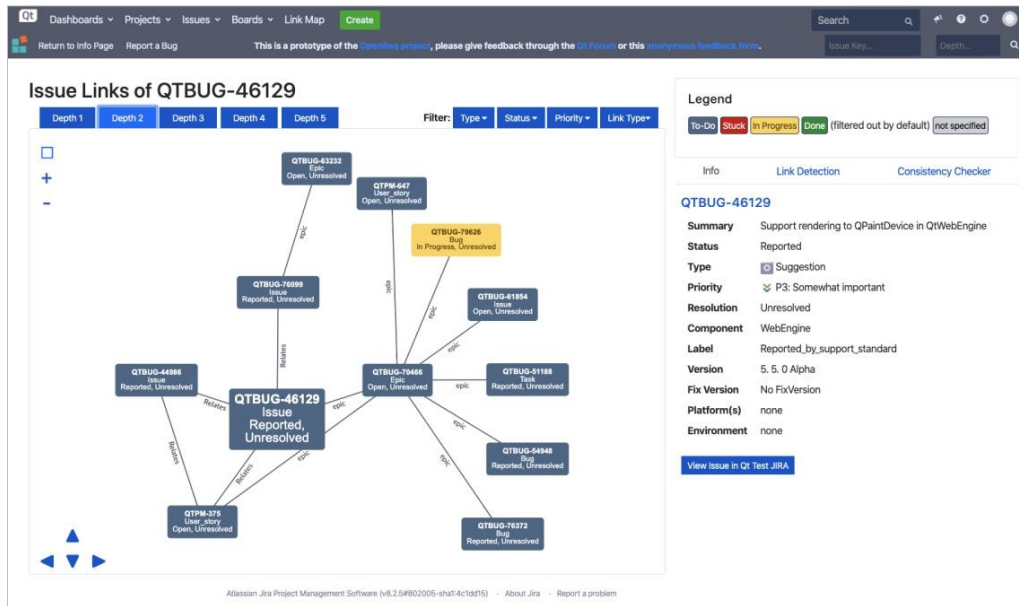


Figure 3.7.: QT plugin visualization of a requirement / issue called "QTBUG-46129"

The results of a usability study of the JIRA plugin show that the provided recommendation functionalities were appreciated by the users. They used the tool differently depending on their role in the company (some appreciated the link detection, others the visualization or the consistency checker). In general, we can summarize that the OPENREQ technologies improved the processes as the developed methodologies make the current status of requirements (especially, incorrect or missing information) more explicit.

Recommenders for the ECLIPSE Community As part of OPENREQ, an ECLIPSE plugin has been developed to support the ECLIPSE open-source community in stakeholder identification tasks. The developed plugin presents a personalized prioritized list of ECLIPSE requirements to an ECLIPSE developer. The underlying concepts and algorithms have been discussed in Section 3.6.1 (see also Samer et al., 2019) and Stanik et al., 2018). In the ECLIPSE world, everything is called a "bug", including actual user requirements, tasks, as well as bug reports. We use the ECLIPSE terminology (bug) in the remainder of this section. The recommendation solution consists of a front-end and a back-end component. The front-end is an ECLIPSE plugin directly included into the ECLIPSE integrated development environment (IDE). This plugin presents a prioritized list of bugs to an ECLIPSE developer taking into consideration the developer's personal preferences, his / her past work, and the priority of the bug for the whole ECLIPSE project. The user has the option to give feedback to the back-end component by rating the suggestion. Furthermore, the user is able to enter dedicated keywords, just in case the user would like to work on topics he or she did not work on (so much) in the past. With the feedback of the developers the back-end learns to provide better results in future recommendations –

the more feedback the users provide, the better the results will be in the future.

Id	Summary	Priority	Product	Component	Creation Date	Disli...	Sno...	Like
550453	Performance loss in Composite.WM_paint()	100%	Platform	SWT	2019-08-26	👍	👎	👍
552832	PDE api-tools in CI build (only CI, not local) report erroneous API Errors	98%	Platform	Releng	2019-11-08	👍	👎	👍
506696	Ctrl+E bugs (Next Editor/View/Perspective switcher popups)	93%	Platform	UI	2016-10-28	👍	👎	👍
547485	IllegalArgumentException using placeholder in model fragments	93%	Platform	UI	2019-05-20	👍	👎	👍
537810	Eclipse Neon does not start after trying to fix missing toolbar	91%	Platform	UI	2018-08-09	👍	👎	👍
485167	[tests] Reactivate disabled UI tests to run regularly with UITestSuite	91%	Platform	UI	2016-01-04	👍	👎	👍
527378	Stop supporting old update manager for launching	91%	PDE	UI	2017-11-17	👍	👎	👍
489335	PartServiceImpl.getActivePart() returns part from the different perspective during p...	89%	Platform	UI	2016-03-10	👍	👎	👍
549295	View added to J2EE perspective does not show after plugin install	88%	Platform	UI	2019-07-15	👍	👎	👍
550967	[10.15] Mac dark theme not listed in Eclipse	87%	Platform	SWT	2019-09-11	👍	👎	👍
540640	Eclipse fails to open non.Java (and sometimes Java) files	87%	Platform	UI	2018-10-31	👍	👎	👍
549802	Re-enable and fix CommandsTestSuite unit tests	87%	Platform	UI	2019-08-05	👍	👎	👍
465456	E4 should not store reference to view and editor icon contribution value	87%	Platform	UI	2015-04-24	👍	👎	👍
537148	[9+] Compiler broken	87%	JDT	Core	2018-07-18	👍	👎	👍
533555	Double click on a file in staged changes will make the other files disappear	86%	EGit	UI	2018-04-13	👍	👎	👍
520282	Get NullPointerException (Only Error) in Product Launch with Validated Plugins with ...	85%	PDE	UI	2017-07-27	👍	👎	👍
547762	osgi.bundles and start levels from existing config.ini are not considered	85%	UI	UI	2019-05-29	👍	👎	👍
551708	[win32] Auto Complete Crashes Eclipse - Eclipse Installer Also Crashes With Same Error	85%	Platform	SWT	2019-10-02	👍	👎	👍
552817	[Dark Theme] Simplify Form styling	84%	Platform	UI	2019-11-08	👍	👎	👍
551163	[Dark Theme] Selectable Form Editor Title should be dark when Eclipse is in dark the...	84%	Platform	UI	2019-09-17	👍	👎	👍
552700	[Dark theme] Use same color for table and tree header background as for the other ...	84%	Platform	UI	2019-11-05	👍	👎	👍
533817	Eclipse UI freeze after idle (eclipse in background or pc sleep)	84%	Platform	UI	2018-04-19	👍	👎	👍
553044	Deadlock in annotation/code mining processing	84%	Platform	Text	2019-11-14	👍	👎	👍

Figure 3.8.: ECLIPSE plugin presenting a personalized prioritized list of ECLIPSE requirements for a developer of the ECLIPSE IDE.

Furthermore, the user has the possibility to see the calculated keywords from the back-end component to understand why the prioritization was done as it was - usually end-users want to understand to a certain degree as to why (and how) the recommendation has been generated / calculated (Felfernig et al., 2007). The following screenshot shows the settings page of the ECLIPSE plugin as well as the calculated keywords in a diagram (see donut chart in Figure 3.9).

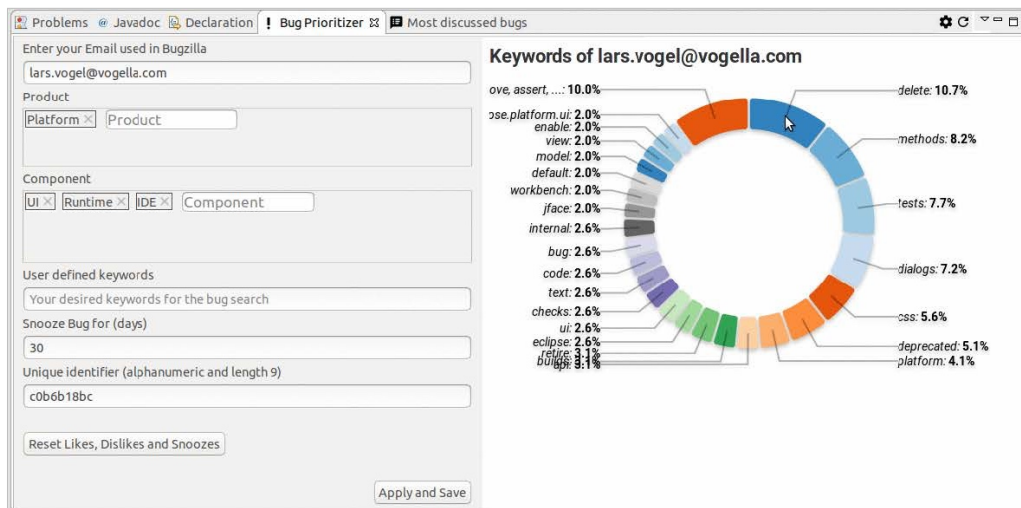


Figure 3.9.: ECLIPSE plugin settings page including calculated keywords.

In general, the back-end component works on the basis of a novel content-based recommendation approach with the capability to adapt to user feedback (see Samer et al., 2019). With the help of the ECLIPSE plugin, ECLIPSE contributors can easily find a bug to work on.

3.7. Selection of Recommendation Algorithms

The variety of different recommendation algorithms discussed in Section 3.5 consists of *content-based filtering* (CB), *collaborative filtering* algorithms (CF), *knowledge-based recommenders* (KR), *group recommenders* (GR), and more sophisticated recommenders based on *supervised classification* (SC) and *clustering* (CL). Depending on the application area, different recommendation algorithms are applicable. Table 3.7 presents general criteria to guide and assist stakeholders in the selection of a recommendation algorithm. The selection criteria are based on the work of Felfernig et al. (2019) and have been adapted to the RE context. The remainder of this section provides a brief discussion of these selection criteria.

Approach	CB	CF	KR	GR	SC	CL
Easiness of setup	✓	✓	✗	✓	✗	✓
Adaptivity	✓	✓	✗	✗	✓	✓
Sparse data	✓	✓	✓	✗	✗	✓
Scalability	✓	✓	✓	✗	✓	✓
Serendipity effects	✗	✓	✗	✓	✓	✗
Cold-start problem	✓	✓	✗	✗	✓	✗
Transparency	✗	✗	✓	✗	✗	✗
High-involvement items	✗	✗	✓	✓	✗	✓

Table 3.7.: General overview of the different recommendation approaches and basic criteria to guide the selection of an algorithm.

Easiness of Setup. There is a large variety of content-based, collaborative filtering, and group recommendation approaches which only require basic setup. The necessary steps to setup RS based on one of these algorithms require the collection of basic details of the items (e.g., if the items represent requirements, then the item details may include the title, the description, and the categories of these requirements) and user / stakeholder preference elicitation on the level of implicit or explicit feedback. In case of clustering-based approaches, items are grouped and separated based on item-related attributes (e.g., requirement type, textual content of the requirement description, categories assigned to a requirement) which also makes this type of recommendation approach easy to setup and use. Classification-based approaches rely on having enough labeled data available to train a prediction / recommendation model, and the setup of these systems is often more time-consuming and challenging, due to data preprocessing and feature engineering efforts. Knowledge-based approaches require more specific and explicit recommendation knowledge (in terms of constraints or semantic properties, e.g., requirement C must be part of release Y) in order to configure these systems.

Adaptivity. The term adaptivity expresses the capability of a RS to easily and automatically consider newly added ratings in future recommendations without any further actions to be triggered. The adaptivity of a RS to new as well as rapidly changing rating data in the context of RE represents

an important selection criteria. The main reason for this is that in many RE-related scenarios (e.g., group bidding-process scenarios (Samer et al., 2018) or the estimation of a software product’s market relevance) rating / evaluation changes of existing ratings as well as new ratings are given by stakeholders throughout the RE process. Therefore, for RS in RE it is important to be able to quickly and seamlessly adapt to new feedback and rating data. Traditional content-based filtering, collaborative filtering, group-based, and clustering-based recommendation approaches are able to easily adapt to new item evaluations (Felfernig et al., 2019) since they can be regarded as learning concepts which belong to the category of *instance-based learning algorithms* (also referred to as *memory-based learning*). Moreover, knowledge-based recommenders require manual changes of the utility schemes, as these schemes are not automatically learned (von Winterfeldt and Edwards, 1986). More sophisticated collaborative filtering algorithms (e.g., matrix factorization) use *model-based learning approaches* to learn a prediction model that generates predictions based on inferences. Likewise, supervised classification algorithms can also be divided into the two types of *instance-based learning* (e.g., k-Nearest Neighbors, support vector machines, etc.) and *model-based learning* (e.g., logistic regression, neural networks, etc.). Instance-based learning algorithms (e.g., k-Nearest Neighbors) can easily and continuously adapt to current changes by simply adding the new samples to the existing knowledge base (i.e., highly adaptive). For *model-based learning* approaches, further training is necessary to automatically adapt learned prediction models. The adaptivity of a model to quickly adapt to newly added instances / ratings primarily depends on whether a model can be trained *offline* or in an *incremental* fashion (*batch / offline training* vs. *incremental / online training*). Algorithms that require *offline learning* have a low adaptivity since new samples / ratings require an effortful retraining of the model using the full dataset. On the contrary, *online learning* usually allows a fast adaption of a model to the new samples / ratings by just retraining the model with the new samples instead of the complete dataset. For example, in case of neural networks (e.g., using mini-batches), the already learned network weights can be used as existing weights to quickly adapt (depending on the used learning rate) the network to the new samples (Perrone et al., 2019).

Sparse Data. In practice, the rating data available to learn a recommendation model is often sparse. This results out of the fact that most users only rate a small set of preferred items in relation to the large assortment of items that exists in an item database (see also Section 3.5.1). This means that most users only evaluate a few items and the vast majority of items remains unrated by these users. In the context of RE, large industrial software projects often consist of up to thousands of requirements. The stakeholders of such projects are usually not able to analyze and evaluate all requirements. This leads to challenging situations for RS in RE where sparse evaluation / rating data is available. Consequently, sparse rating data is a challenging issue for many RS in RE. In particular, sparse rating data poses a challenge for group-based and classification-based recommendation models, since such models require a decent amount of rating data available for every item. In particular, group-based recommenders using heuristics to generate prioritized sequences of items / requirements require preference elicitation

of all users which should be as complete as possible. Knowledge-based recommenders require more detailed information about a user's preferences and the constraints in order to discover and recommend appropriate solutions. Classification-based recommenders need a more dense amount of training samples (which convey sufficient details about all individual user preferences) and a well-balanced set of training samples (i.e., enough samples for each class). Content-based filtering recommender systems are able to handle sparse rating information quite well as long as sufficient amount of content-related information of the items (attributes or categories) is available (Idrissi and Zellou, 2020). In case of neighborhood-based collaborative filtering approaches, the prediction quality depends on the number of ratings provided by other / similar users. This means that if an item has not been rated by many users, then it cannot get recommended often. However, model-based collaborative filtering systems (e.g., matrix factorization) represent a good and efficient alternative to solve this problem (Idrissi and Zellou, 2020).

Scalability. An important requirement to a RS is its *scalability*. In many practical RE-related scenarios (Mobasher and Cleland-Huang, 2011), the number of items (e.g., requirements) is very large, which significantly increases the complexity of building / learning a prediction model. Content-based filtering and item-based collaborative filtering solutions are capable of handling large numbers of users / stakeholders, items, and item ratings in an efficient manner. However, *model-based learning approaches* (e.g., matrix factorization) usually perform more efficiently on large datasets (e.g., lower memory consumption and prediction effort) when compared to *instance-based learning* algorithms (basic content-based filtering or user-based collaborative filtering). *Model-based learning* also helps to effectively counteract noise in the rating data. Moreover, knowledge-based and clustering-based recommendation approaches are also able to scale with an increasing number of items and users. In case of classification-based recommenders, deep learning architectures can be used to even take advantage of the full knowledge power of extremely large datasets (Goodfellow et al., 2016).

Serendipity Effects. The *serendipity effect* describes the event of an unexpected and fortunate discovery of new items by a recommender system. It expresses a surprising moment when relevant items are proposed to a user which the user did not expect to see. In practice, such items can be niche products or in RE, these items can be requirements that are of high relevance for a stakeholder but related to subsidiary domains (or relevant requirements that have not yet been thought about by the stakeholder). Content-based approaches are based on the principle of recommending content-related items of previously liked items and are therefore not suitable to discover new unexpected items that do not relate to previously seen items (in terms of the content). Similar to content-based filtering, the concept of clustering-based recommendation also follows an approach of bringing together similar items (based on some characteristics or the content). The items recommended (or generated) by knowledge-based recommenders conform to specified constraints, recommendation criteria, and user preferences. In order to achieve *serendipity capabilities*, the existing knowledge base of knowledge-

based recommenders has to be enriched with specific serendipity settings. In contrast, collaborative filtering recommenders are appropriate solutions to find and unveil unexpected items, since such systems follow the approach of finding new items preferred by like-minded users. Hence, collaborative filtering has *serendipity capabilities* which allow to discover new items that are not related to the content of any other item liked by the current user, but still refer to the profile of the current user (Kotkov et al., 2016). Serendipity effects can also be observed when using supervised classification recommenders, as these systems learn latent patterns in the data in order to generate recommendations. Moreover, heuristics have to be adapted to allow serendipitous recommendations in group-based recommendation settings (Masthoff, 2015).

Cold-start Problem. The cold-start problem describes a situation where no or only very sparse information about users (*user cold-start problem*) or items (*item cold-start problem*) is available. In practice, such situations can occur, for example, at the beginning when a recommender system is used for the first time, or during operation when new items or users are added to the system. Unlike most general RS (e.g., movie recommenders), RS used in RE have to face different difficulties and challenges of the cold-start problem. In the context of RE, the requirements often represent the items in a typical RE-related recommendation scenario and there exists a large number of requirements which have not been evaluated at the beginning. In case of movie recommender systems, new items / movies are typically continuously added over a long period of time. However, RS used in RE have to deal with situations where only a set of unrated items / requirements is available at the beginning. Consequently, in the context of RE, the item cold-start problem is usually more relevant than the user cold-start problem. Collaborative filtering recommenders are prone to cold-start issues as they require a sufficient amount of rating data. In contrast, content-based recommenders can handle new items quite well since the algorithm can make associations between existing user profiles and new item data without any new ratings. However, *user-related cold-start problems* remain an issue for content-based recommenders. Similar to content-based algorithms, knowledge-based approaches also represent a suitable solution to tackle item-related cold-start issues. Hybrid recommendation approaches are in most cases a good chance to tackle the typical cold-start problems.

Transparency. In terms of RS, the degree of transparency indicates how explainable the recommendation results are. Explainable recommendations are important to increase the trust of the users in a RS. In the context of RE, explainable and transparent recommendations are fundamental as the recommendations often provide the basis for project-critical decisions (e.g., requirements triage or release planning) and the explanations of these recommendations then allow the stakeholders to justify their decisions. Since content-based systems recommend items that are similar to items previously consumed by the current user, recommendations can be explained by describing this principle (e.g., the following items have been recommended because the items that you have purchased in the past are similar to these recommended items). Recommendations generated by collaborative systems can

often be explained based on the similarity between the current user and the other neighbor users from who the recommended items have been taken / extracted (e.g., *"these requirements have been recommended for evaluation because like-minded stakeholders evaluated these requirements."* or *"most stakeholders who have seen this requirement also have viewed the following requirements..."*). Even though these explanations of content-based and collaborative filtering systems seem to be quite simple and easy-to-understand, the explanations can not be regarded as being transparent as they do not allow the user to obtain deep insights to the reasons of the recommendations. Likewise, explanations of recommendations generated by systems based on supervised classification or clustering can also be regarded as being shallow as these systems also represent a specific form of content-based or collaborative filtering systems on an abstract level. In contrast, explanations of items recommended by knowledge-based recommenders provide a high degree of transparency since the explanations represent information that was gained during the reasoning process (Felfernig and Burke, 2008). The explanations of recommendations delivered by group-based systems are also often transparent. However, the explanations primarily depend on the used heuristics (aggregation functions) to calculate the utility value of the items (see Section 3.5.2 and Felfernig et al., 2018).

3.8. Open Research Topics

The RE process is composed of different activities where each of them involves several tasks. Although there exist many recommendation tools which already cover and support many of these tasks, we can still observe some gaps regarding decision-support in RE that should be addressed by future work.

Prediction Quality. For example, future work should address the improvement of the efficiency and the prediction quality of recommendation approaches applied in RE. This helps to keep up with the upcoming increase of challenges introduced by the ongoing growth of software complexity. *Deep learning* (Goodfellow et al., 2016) techniques are currently emerging to become an important means for RE (Baker et al., 2019; Navarro-Almanza et al., 2017). Such techniques can empower RS to further increase the prediction and decision quality by exploiting large data sources. In case of large software projects, the relevant information to support important decisions can be qualitatively enriched by discovering vast amounts of data (e.g., user feedback from large social media channels). For example, the application of more sophisticated *deep neural networks* and *deep convolutional neural networks* as document classification solutions in large industrial RE projects, can help to significantly increase the performance of correctly classified requirements (defined in natural language). Moreover, existing work (e.g., Dekhtyar and Fong, 2017) demonstrates the potential that pre-trained word embeddings (Goodfellow et al., 2016) can help to achieve higher prediction rates in specific requirement classification tasks. However, more research has to be conducted, in order to apply such techniques and further improved versions of existing solutions in more RE-tasks (such as requirement

dependency detection, requirements reuse, requirements prioritization, requirements quality analysis, etc.). Moreover, the integration of these techniques into requirements management platforms and recommendation tools also represents an important part for future work.

Datasets. The aforementioned techniques are based on supervised classification and require a large amount of highly qualitative labeled training data (i.e., a solid *ground truth*), in order to build reliable and correct recommendation models. Although there exist collections of public RE datasets (e.g., Ferrari et al., 2017) from many different domains (web development, telecommunication, railway, etc.), further data collection and manual processing is necessary for most of these datasets in order to gather a sufficient amount of high-quality data that is suitable for training and evaluating prediction models based on supervised classification (*deep learning* as well as *traditional supervised classification approaches*). In particular, in the field of automated requirement dependency detection, more comprehensive, diverse, and high-quality requirement dependency datasets are required to allow more expressive as well as representative studies and evaluations of different dependency identification approaches as well as reasonable comparisons between existing solutions. For example, existing dependency detection approaches which are based on supervised classification (Samer et al., 2019; Deshpande et al., 2019) have only been evaluated on relatively small datasets which stem from one single or a few specific domains. Moreover, we can also observe a need for larger datasets from different domains that contain evaluation / rating data of requirement evaluations conducted in groups. This can help to quickly construct evaluation baselines against which different group recommendation approaches can be compared and define the basis for refined approaches that can increase decision-making in group-based evaluation settings. Ongoing work will address these issues and new datasets for particular RE-related recommendation tasks are planned to be published along with improved solutions.

Explanations. The presentation of explanations related to recommendations in RE-related tasks represents another aspect on which future work should focus on. To the best of our knowledge, we can observe that this represents an aspect where the RE community has still a huge potential for improvements. Explanations help stakeholders to understand how certain recommendations were generated by a RS which usually leads to more trust and to a reduced level of stakeholder uncertainty in the decision-making process (Tintarev and Masthoff, 2007). For example, visualization tools with (visual and textual) explanations can be applied in order to show more detailed statistics that visualize requirement evaluations and deviations of diverging opinions. Such explanations can trigger discussions which represent a prerequisite for improved RE-related decisions (in particular, requirements prioritizations).

Gamification. Future work regarding requirements management platforms (such as OPENREQ!LIVE), should take concepts from gamification (Sailer et al., 2017) into account, in order to increase the en-

agement of stakeholders in RE decision-making. Gamification is known as the application of principles in the fields of *game design*, *game dynamics*, *behavioral economics*, and *motivational psychology* in software platforms. The gamification concept exploits basic human behaviors such as impatience, curiosity, fear of loss, and social influence to design software that focuses on users (Chou, 2019). In general, we can observe that gamification techniques are very successful in software systems today (Chou, 2019; Sailer et al., 2017). Modern online platforms (such as Facebook¹⁸, LinkedIn¹⁹, or Instagram²⁰) use these principles to motivate users to take specific actions. Examples of the successful application of gamification mechanisms include, for example, status indicators indicating the completeness of the user profile on a business platform using the so-called "Need for Completion" behavior pattern. Lombriser et al. (2016) show that gamification can positively influence the elicitation process in agile RE. The authors found that simulating competitions with the help of gamification can help to collect basic as well as novel requirements, and gamification has a significant positive impact on creativity. The successful application of gamification techniques heavily depends on the selection of game elements, as they can influence different psychological aspects. The conducted experiment of Lombriser et al. shows that an individual leader-board and the possibility to win prizes encourages the competition in a positive manner. Rivalries of different stakeholders increased requirements production and thus resulted in a higher quality as well as more creative ideas (Lombriser et al., 2016). Due to the fact that most requirements engineering processes are tasks for groups of users, decision biases are very likely to arise and thus can lead to significantly lower decision outcomes (e.g., release plans). Hence, an interdisciplinary research will be needed to take into account related decision psychological theories (Stettinger et al., 2015).

Sustainability. Global warming represents one of the greatest challenges in our human history and measures to combat global warming will affect all areas of our daily life. Regarding future work in the field of RE, the focus should also lie on the development of sustainable requirements, in order to develop systems that are as resource-efficient as possible. Future research should also address this topic in order to contribute to this global human undertaking. In particular, more recommendation tools are needed that foster sustainability aspects in the planning and development of complex sustainable software-driven solutions (e.g., solutions for complex smart city projects).

3.9. Conclusion

This chapter gives an overview of different recommendation approaches applied in the context of *requirements engineering* (RE). The RE process can be viewed as a complex decision-driven process consisting of many different phases in which many stakeholders are usually involved. A high complexity in the process also implicates a high risk of project failure. As a consequence, there is a high

¹⁸Facebook: <https://www.facebook.com>

¹⁹LinkedIn: <https://www.linkedin.com>

²⁰Instagram: <https://www.instagram.com>

demand for intelligent decision-support tools which can help to limit and reduce this risk. The variety of intelligent recommendation techniques presented in this chapter fits seamlessly into this problem. It ranges from basic recommendation-based methods which focus on decision support to more sophisticated recommendation techniques that assist groups by decreasing the information overhead in order to improve decision-making. We have discussed different types of recommendation algorithms and provided several basic examples to demonstrate some of the underlying algorithmic approaches. In this chapter, we focused on a discussion of RS in RE and showed how innovative recommendation concepts have shaped RE research in recent years. Thereby, we presented a series of recommendation tools which are applied in different phases of the RE process. Although the general results produced by such recommendation tools are still far away from results which can be achieved by humans, the results are good enough to provide high qualitative suggestions to stakeholders. Throughout the chapter, we have also provided a brief insight into the OPENREQ project which represents an example of a large research project in the field of recommendation-applied RE. Within the scope of the project, innovative recommendation technologies for the area of RE have been developed. Relevant evaluation results of selected representative studies conducted by the project partners have been presented in this chapter to demonstrate the high potential of these recommendation tools to foster and improve decision-making in RE. Moreover, we introduced the requirements management platform OPENREQ!LIVE which has been developed within the scope of the OPENREQ project. The platform features a rich set of intelligent decision and recommendation techniques which support various RE tasks (reaching from requirements elicitation over stakeholder assignment till complete release planning of complex software projects). The OPENREQ project is important, not only for tool support, but its case studies can guide research in directions that are supported by actual real-world cases from people who are involved in the field of RE. Finally, with our outlook on topics regarding future work in this area, we intend to motivate the development of more refined recommendation tools in RE which satisfy emerging industrial needs.

New Approaches to the Identification of Dependencies between Requirements

This chapter is based on the results documented in Samer et al. (2019). Most parts of this chapter, such as literature research, algorithmic recommendation approaches as well as the user study have been provided by the author of this thesis.

4.1. Abstract

There is a high demand for intelligent *decision support systems* which assist stakeholders in requirements engineering tasks. Examples of such tasks are *the elicitation of requirements*, *release planning*, and *the identification of requirement dependencies*. In particular, the detection of dependencies between requirements is a major challenge for stakeholders. In this chapter, we present two content-based recommendation approaches which automatically detect and recommend such dependencies. The first approach identifies potential dependencies between requirements which are defined on a textual level by exploiting document classification techniques (based on *Linear SVM*, *Naïve Bayes*, *Random Forest*, and *k-Nearest Neighbors*). This approach uses two different feature types (*TF-IDF features* vs. *probabilistic features*). The second recommendation approach is based on *Latent Semantic Analysis* and defines the baseline for the evaluation with a real-world dataset. The evaluation shows that the recommendation approach based on *Random Forest* using probabilistic features achieves the best prediction quality of all approaches (f1-score: 0.89).

4.2. Introduction

Recommender systems (RS) are decision support systems which help users to select a well-collected set of items matching their needs and preferences (Adomavicius and Tuzhilin, 2005; Resnick and Var-

ian, 1997). Nowadays, these systems are applied in many well-known domains such as books, movies, or songs. In more complex domains such as *requirements engineering* (RE), there is a high demand for applying RS to support stakeholders (Felfernig et al., 2013; Mobasher and Cleland-Huang, 2011). Recommender systems can support stakeholders in different RE tasks such as *requirements definition / elicitation, release decisions, stakeholder identification, and dependency detection* (Mobasher and Cleland-Huang, 2011; Ninaus et al., 2014).

Usually, a software project consists of hundreds of different requirements which are often related to each other. Single requirements elicitation methods such as interviews (Davis et al., 2006) are considered effective, but do not scale up for the elicitation of dependencies. The identification of dependencies between the requirements is a cognitively challenging and time consuming task which requires the use of intelligent methods (Leffingwell, 1997; Mobasher and Cleland-Huang, 2011). The traditional method, where stakeholders detect requirement dependencies manually, entails a high risk of project failure, since stakeholders are often not aware of the latest changes regarding the set of requirements. In addition, stakeholders have to understand the domain-specific content of each requirement which is also very time-consuming. Missing or incorrect dependencies will result in release plans that require additional effort for their implementation (Ruhe, 2010). There exist different types of requirement dependencies such as *includes, excludes, and requires*. In particular, *requires* is known to be the most frequently occurring dependency type in the context of RE (Ferber et al., 2002). The identification of *requires*-dependencies is essential for a project, because the late discovery of these dependencies can lead to negative consequences such as increased costs or unfulfilled deadlines. In order to increase the quality of software release planning, more sophisticated approaches are needed. Therefore, we developed two content-based recommendation approaches which support stakeholders in the identification of such dependencies.

In the context of dependency recommendation, there exists some related work (Carlshamre et al., 2001; Deshpande, 2019). The work of Chitchyan and Rashid (2006) describes an NLP-based (*natural language processing*) approach which assists in the identification of dependencies between requirements on a semantic level. Ninaus et al. (2014) present an RE tool which applies recommendation techniques to support stakeholders in RE tasks. Their tool also includes a basic dependency recommendation approach which recommends similar requirements that are treated as potential dependencies. Atas et al. (2018) presents an approach to automatically identify requirement dependencies of type *requires* by using supervised classification techniques. Having a high prediction quality is crucial for effectively supporting stakeholders. Our major research goal is to further improve the prediction quality of dependency detection compared to existing approaches. Our recommenders follow the objective to provide decision support to domain experts in the task of dependency elicitation. The major contributions of this chapter are the following. The work presented in this chapter extends the basic approach of Atas et al. (2018) and enriches it with new feature types and a new classification approach

based on aspects from the area of *information theory*. In contrast to Atas et al. (2018) and Ninaus et al. (2014), our work uses (1) enhanced methods that achieve higher prediction quality and (2) a recommendation environment based on the developed classification approaches. We also provide a new dataset which can be used as baseline for related comparisons. The evaluation results indicate that our developed approaches are able to reliably identify dependencies between requirements. In particular, the results reveal that our content-based RS based on *Random Forest* classification achieves a high prediction quality.

The remainder of this chapter is structured as follows. In Section 4.3, we explain the structure of the used dataset and the design of an empirical study to manually detect the requirement dependencies included in the dataset. Section 4.4 presents the used pre-processing and feature extraction techniques. In Section 4.5, we introduce approaches to automatically detect and recommend dependencies between requirements. An experimental evaluation of our content-based RS and a short discussion of the evaluated results is provided in Section 4.6. Finally, we conclude the chapter and provide a brief outlook towards future work in Section 4.7.

4.3. User Study & Dataset

We used a dataset¹ which consists of 30 software and hardware requirements as well as of 51 dependencies that exist between these requirements. The requirements were related to the development of a sports watch and have been defined in cooperation with software development companies (industry partners). The industry partners are experts with longstanding experience and practical knowledge in the RE domain. Each of the defined requirements consists of an *id*, a *title*, and a textual *description* (in German). We conducted a user study in a software engineering course with N=182 computer science students and asked them to manually detect dependencies of type *requires*. A *requires*-dependency for an ordered requirement pair (r_x, r_y) is a unidirectional dependency which indicates that r_x requires r_y (denoted as: $r_x \rightarrow r_y$). This statement does not imply that r_y also requires r_x . However, it is important to mention that our work presented in this chapter only focuses on the prediction of a *requires*-dependency but not on the prediction of its direction.

The major aim of our user study was the complete detection of all dependencies for the predefined set of requirements. For the purposes of the user study, a set of 30 requirements was presented to each participant. In order to avoid biasing effects (Murphy et al., 2006), a randomly ordered list of these 30 requirements was shown to each participant. The identified dependencies were used for training and testing of our content-based RS. Considering the direction of the *requires*-dependencies, the number of possible dependencies is $\binom{30}{2} * 2 = 870$. In order to obtain a complete dataset and a profound ground truth basis to train our RS, we reviewed and combined the most frequently reported

¹Dataset: http://openreq.ist.tugraz.at:8080/OpenReq_dataset.zip

dependencies with the dependencies of an example solution from 7 experts of our industry partners. We cleaned the dataset in collaboration with these RE experts in order to train the RS with the correct dependencies. Thereby, also less frequently reported (but correct) dependencies could be found and were included in the final dataset. This way, a complete dataset could be derived which represents a ground truth that assures completeness, preciseness, and clearness of the data.

Table 4.1 provides a brief overview of the dependencies reported by the experts and the study participants. The study results show that the 182 participants stated 657 different dependencies. Our 7 experts stated 38 dependencies and found more unique dependencies (5.43) on average than the participants (3.61). In order to obtain a final solution from the collected data, we took the 20% of the students' most frequently reported dependencies (131) and combined them with the 38 dependencies reported by the experts. Considering the intersection of both sets, there was an overlap of 35 dependencies. We analyzed (together with the experts) the remaining part of the non-overlapping dependencies $((657 - 35) + (38 - 35) = 625)$ reported by the students and the experts. This way, another 16 dependencies could be obtained which were added to the set of 35 overlapping dependencies. The final dataset consists of 51 *requires*-dependencies $(35 + 16 = 51)$ and 30 requirements.

		Reported Dependencies	
Group	Persons	Amount	Average
Study Participants	182	657	3.61
Experts	7	38	5.43
Overlap		35	-
Additionally added		16	-
Finally selected		51	-

Table 4.1.: Dependencies found by experts and students.

4.4. Preprocessing & Feature Extraction

Before the system could be trained and tested, the records of the final dataset had to be prepared and converted into a format which is suitable for a recommender system based on machine learning. For the preprocessing of our dataset, we first tokenized the title and the description of each requirement into proper linguistic units (*bag of words*). Thereafter, we removed stop words and special characters, merged synonyms, and applied lemmatisation.

4.4.1. Extraction of TF-IDF Features

For every token the *term frequency-inverse document frequency* (TF-IDF) was determined. After the calculation of the TF-IDF values, tokens which do not contain any valuable information were

removed. In our approach, the TF-IDF value was calculated for single (i.e., uni-gram) and adjacent tokens (i.e., n-grams). Then, TF-IDF values were combined into a single vector \mathbf{v} for a requirement pair (r_x, r_y) . Formula 4.1 provides a formal representation of the feature vector \mathbf{v} whereby each TF-IDF value of an n-gram token is wrapped in a separate mathematical set (see curly brackets) and these sets are then merged with each other by using the union operator "∪".

$$\mathbf{v}(r_x, r_y) = \bigcup_{i \in \text{ngrams}(r_x)} \{TFIDF(i)\} \cup \bigcup_{j \in \text{ngrams}(r_y)} \{TFIDF(j)\} \quad (4.1)$$

4.4.2. Extraction of Probabilistic Features

As an alternative feature representation, we also used features which take aspects from the area of *information theory* into account (Durme and Lall, 2009). In the remainder of this chapter, we call these features *probabilistic features*. We used probabilistic features as an alternative to TF-IDF features, since they reflect statistical correlations between the words and provide more precise descriptions of the word-based similarity of the requirement pairs. For each pair (r_x, r_y) , we created features that express correlations between the words from the title and the description of a requirement. We created features which are based on the co-occurrences of words. These features are counted values that reflect the number of words which both requirements share in common. Further, we also introduced probabilistic values as additional features. The probabilistic values are computed by using the *Pointwise Mutual Information* (PMI) measure (see Formula 4.2).

$$pv(T_x, T_y) = \sum_{w \in T_x} \sum_{v \in T_y} \log \frac{p(w, v)}{p(w) * p(v)} \quad (4.2)$$

The token list of requirement r_x (T_x) and requirement r_y (T_y) were compared and $pv(T_x, T_y)$ was determined by summing up the PMI values of all possible token-pairs among T_x and T_y . The following features were used:

- **feat1:** overlap between description-tokens of r_x and all tokens of r_y
- **feat2:** overlap between description-tokens of r_y and all tokens of r_x
- **feat3:** PMI of title-tokens of r_x and r_y
- **feat4:** PMI of description-tokens of r_x and r_y
- **feat5:** PMI of all tokens of r_x and r_y

Feature *feat1* refers to the number of description tokens of r_x that also occur in the list of title-tokens or description-tokens of r_y . In other words, we quantify the absolute value of the word-overlap for a given requirement-pair (r_x, r_y) between those tokens that appear in the description of r_x and those tokens which appear in the title or description of r_y . Likewise, feature *feat2* corresponds to the counted value reflecting the number of those description-tokens of requirement r_x that also co-occur in the list of description-tokens of r_y . In addition to these two features, we introduced three probabilistic features which were all calculated based on Formula 4.2. The idea of these features consists in measuring the probability for each word-pair among T_x and T_y that the word / token $w \in T_x$ and the token $v \in T_y$ co-occur (i.e., $p(w, v)$) in relation to the individual probabilistic occurrence of w (i.e., $p(w)$) and the individual probabilistic occurrence of v (i.e., $p(v)$). To compute the value of a requirement-pair (r_x, r_y) for feature *feat3*, the title-token list of r_x (denoted as T_x) and the title-token list of r_y (denoted as T_y) are compared with each other. The PMI value for each token-pair among T_x and T_y is individually calculated and summed up. The sum of all PMI values is then used as feature *feat3* for the pair (r_x, r_y) . Likewise, the values for feature *feat4* and feature *feat5* can be obtained by using the same procedure. In case of *feat4*, the token list T_x is replaced with the *description*-tokens of r_x and the token list T_y only contains the *description*-tokens of r_y . For *feat5*, T_x is considered as the list of all tokens of requirement r_x and T_y consists of all tokens of requirement r_y . Once all five features for a pair (r_x, r_y) are obtained, a feature vector can be constructed. Before the feature vector is passed as input to the classifier, feature scaling is applied to each vector component individually. This ensures that every feature value $u \in \{feat1, feat2, feat3, feat4, feat5\}$ is normalized and equal importance is given to all features.

4.5. Approach

We developed two different content-based recommendation approaches which follow the objective to identify and recommend dependencies between requirements. Both approaches were trained with a training set (TR) and tested with a test set (TE). Our implementation was based on the *Scikit-learn library*².

4.5.1. Classification (Approach I)

Our first RS used a binary classifier to predict the existence of a dependency (*true* vs. *false*). We compared different classifiers based on *Linear SVM (Support-Vector Machine)*, *Naïve Bayes*, *Random Forest*, and *k-Nearest Neighbor (k-NN)* and evaluated their performance (see Section 4.6). We considered all requirements from the training set TR and created training pairs which were used as training samples. Each training sample and each test sample corresponds to a feature vector of a requirement pair (r_x, r_y) and contains either the combined *TF-IDF features* or *probabilistic features* of r_x and r_y . The training pairs / samples were then used to learn a prediction model. To generate all training pairs,

²Scikit-learn library: <https://scikit-learn.org>

we created all possible $(n - 1)$ requirement-pairs for a requirement $r_a \in TR$ with every other requirement $r_b \in TR$. For each pair (r_a, r_b) the feature vector v_{r_a} of r_a was combined with the feature vector v_{r_b} of r_b into a single feature vector v_{r_a, r_b} . The resulting feature vector was then used as a training pair. In the case that a pair (r_x, r_y) was dependent (i.e., $r_x \rightarrow r_y$ and / or $r_y \rightarrow r_x$), we assigned the *true* class to this sample (label=true), otherwise *false* (label=false). Due to a significant class imbalance between the *false* and *true* class, not all training pairs could be used to train the classifier. The number of independent pairs completely dominated the pairs where both requirements were dependent on each other. Thus, we randomly under-sampled the *false* class.

When we used *TF-IDF features*, the feature vector v_{r_a, r_b} contained the TF-IDF values of all tokens that occur in the title and description of r_a and r_b (Section 4.4.1). In the case of *probabilistic features*, we used the five computed features described in Section 4.4.2. The classifier was trained with the training pairs by using their feature vector and label. The learned prediction model was then used to predict *dependent* requirements for a given requirement $r_x \in TE$. This was achieved by considering all possible test samples / pairs (r_x, r_y) , where $r_x \in TE \wedge r_y \in TR$. These pairs were passed as input to the classifier. The classifier then individually predicted the existence of a dependency between both requirements for each pair. All those pairs for which the classifier predicted *true* were finally recommended.

4.5.2. Latent Semantic Analysis (Approach II)

In order to compare our RS based on classification, we developed a second recommender which is based on *Latent Semantic Analysis* (LSA) and acts as baseline for the evaluation. LSA utilizes *Singular Value Decomposition* (SVD) to transform a *term-document matrix* into its semantic-space representation (Deerwester et al., 1990). Given the TF-IDF values of all preprocessed title- and description-tokens of the requirements, we created a document-term matrix X . Each training sample and each test sample corresponds to a feature vector of one requirement and contains the TF-IDF features. The idea of this approach consists in building a document-term matrix with the requirements from the training set (80%) and then to use LSA to find requirements which are similar to a requirement r_x from the test set (20%). All such similar requirements are considered as being dependent on r_x and are then recommended as a *requires*-dependency.

In the document-term matrix X , the requirements represent the documents. The preprocessed title and description tokens of a requirement are combined into a single document-vector. This vector contains the tokens' TF-IDF values and appears as a column in X . X is a $m \times n$ matrix where each column j represents the document-vector d_j of requirement r_j . Each row i of X corresponds to a single token / term (t_i^T) . After the construction of X , LSA is applied to decompose X into three matrices U , Σ , and V^T such that the product of these decomposed matrices leads back to the original matrix X (i.e., $X = U\Sigma V^T$).

The three components (U, Σ, V^T) constitute a semantic representation of X . U is a $m \times l$ matrix which maps the terms of the matrix X onto l characteristic features. Likewise, V^T is a $l \times n$ matrix mapping the requirements of the matrix X onto l characteristic features. Σ is a $l \times l$ diagonal matrix where each diagonal entry σ_i represents a singular value. Each singular value refers to the weight / importance of the corresponding characteristic feature. The lowest singular values are removed from Σ due to their low importance and only the k highest ones are preserved. The number of U 's columns and the number of V^T 's rows is also reduced to k . This leads to a truncated semantic representation (U_k, Σ_k, V_k^T) which only contains the k most important characteristics. This way, the dimensionality of the data is reduced and noise is removed which leads to an implicit merge of terms that share similar meanings (e.g., synonyms).

Although also some valuable information gets lost after data reduction, the most valuable information of the original semantic representation (U, Σ, V^T) still remains part of the truncated semantic representation (U_k, Σ_k, V_k^T) . The product $U_k \Sigma_k V_k^T$ still results in a matrix that is quite close to the original matrix X and can be considered as a good approximation of X . The truncated semantic representation (U_k, Σ_k, V_k^T) is used to find requirements similar to a given requirement r_x . This is achieved by transforming the document-vector \mathbf{d}_x of r_x into its reduced semantic space representation \mathbf{d}'_x (see Formula 4.3).

$$\mathbf{d}'_x = \Sigma_k^{-1} U_k^T \mathbf{d}_x \quad (4.3)$$

The transformed document-vector \mathbf{d}'_x of r_x is then compared with all other semantic document-vectors which represent the other requirements in the low-dimensional space. In order to find the requirements that are most similar to r_x , we measure the cosine similarity between \mathbf{d}'_x and all other semantic document-vectors which are the column vectors of V_k^T . The underlying assumption is that the most similar requirements can be considered as being probably dependent on requirement r_x . These requirements are then recommended together with r_x as *requires*-dependencies.

4.6. Evaluation & Discussion

To evaluate both approaches described in Section 4.5, we used *k-fold cross validation* ($k = 10$). The overall prediction quality of the recommended dependencies was measured in terms of *precision*, *recall*, and *f1-score*. In the case of the first RS based on classification, TF-IDF values of unigrams, bi-grams, and tri-grams were used, for LSA (our second RS) only unigrams were considered (see Section 4.4). The classifier of the first RS returned a probability value for each prediction. We used this probability value to limit the number of recommended *requires*-dependencies. We introduced a threshold of 65% such that only predicted dependencies which had a probability above this threshold were

recommended. Likewise, we introduced another threshold parameter for the other approach based on LSA. This threshold parameter was set to 0.8³ and it referred to the minimal cosine similarity that another requirement $r_y \in TR$ must have in order to be considered as being dependent on r_x . The dependencies of those requirements which satisfied the similarity-threshold were finally recommended.

Table 4.2 presents the evaluation results of the different algorithms. According to these results, all algorithms performed quite well with TF-IDF and probabilistic features. This means that the *requires*-dependencies between the requirements were found quite reliably by all algorithms. In particular, high scores in terms of precision and f1-score can be observed for *Linear SVM* (precision: 0.997, f1-score: 0.695) and *Naïve Bayes* (0.901, f1-score: 0.720) when TF-IDF features were used. However, although the recommender using *Linear SVM* classification and TF-IDF features was able to accurately predict dependencies (high precision), it also shows a low recall of 0.533 which indicates that many dependencies could not be found and hence were never recommended by this classifier. This further indicates that the used probability threshold of 0.65 (see Section 4.6) is too high for this classifier and more dependencies with a lower probability should be included in the recommendation list. Moreover, it is noticeable that the LSA approach (which represents the baseline of our evaluation) shows a low precision of 0.6 (i.e., not so many recommended dependencies were correct) but a high recall of 0.818 (i.e., most of all existing correct dependencies were found and recommended). The same also applies to k-Nearest Neighbors (precision: 0.611, recall: 0.733). In case of LSA, this might be due to the reason that the LSA approach can be considered to be acting like a fuzzy clustering algorithm which tends to recommend all requirements that are very similar on a content-based level. This way, most dependencies can be found. However, LSA’s low precision indicates that this approach seems to lack of naïvety and can not develop a good sense to distinguish between those requirements that are just similar versus those requirements that are really dependent.

Algorithm	TF-IDF Features			Probabilistic Features		
	P	R	F1	P	R	F1
LSA (Baseline)	0.600	0.818	0.692	–	–	–
Naïve Bayes	0.901	0.600	0.720	–	–	–
Linear SVM	0.997	0.533	0.695	0.812	0.567	0.668
k-Nearest N.	0.611	0.733	0.667	0.786	0.733	0.759
Random Forest	0.889	0.533	0.667	0.929	0.867	0.897

Table 4.2.: Scores of the different algorithms (precision [P], recall [R], and f1-score [F1]). The highest scores are highlighted.

³We tested different threshold-combinations and achieved the best prediction results with 65% for probab. threshold and 0.8 for dist. threshold.

Since LSA requires a document-term matrix as input, it cannot be combined with our probabilistic features. *Naïve Bayes* is also supposed to be used only with TF-IDF features (or term frequencies). Hence, we could only evaluate the other three classifiers with our probabilistic features. By comparing the previously discussed results (obtained by using TF-IDF features) with the results obtained by using probabilistic features, one can observe a remarkable increase of the prediction quality for all three classifiers (except *Linear SVM*). This is especially true with respect to all measures for the *Random Forest* classifier which achieved the best overall prediction quality (precision: 0.929, recall: 0.867, f1-score: 0.897) and could even significantly outperform the baseline approach in terms of recall. This behavior can be explained by taking a look at the probabilistic feature generation approach. The idea of probabilistic features consists in measuring the co-occurrence of words that appear in two requirements. This ensures that "noisy" words of a given requirement which are unlikely to co-occur in the context of another requirement, are considered as unimportant and hence do not contribute much to the calculated feature values. However, the valuable words of a given requirement that co-occur in the context of another requirement, represent valuable information and lead to a significant contribution to the calculated feature values. Consequently, a more descriptive feature representation containing valuable information can be provided to the recommender based on *Random Forest*. This empowers the classifier to more accurately detect whether or not a dependency between two requirements exists.

4.7. Conclusion & Future Work

Conclusion. In this chapter, we introduced two recommender systems (RS) for the recommendation of requirement dependencies of type *requires*. We focused on the type *requires*, as this type can be considered as the most critical type among all existing dependency types (Ferber et al., 2002). The first RS was based on classification and the second was based on *Latent Semantic Analysis* (LSA). The used classifiers (*Naïve Bayes*, *Linear SVM*, *k-NN*, *Random Forest*) of the first approach were fed first with *TF-IDF features* and afterwards with *probabilistic features*. In contrast to that, only TF-IDF features were used for LSA. The results obtained with TF-IDF features provide a clear indication that all classifiers (except k-NN) achieve a high precision rate. However, LSA (our baseline approach) shows a low precision which is due to its similarity-based approach that tends to find similar requirements instead of requirements which really dependent on a given requirement. Moreover, the analysis reveals that *Random Forest* achieved the best overall prediction quality with *probabilistic features* in terms of all three measures (precision: 0.929, recall: 0.867, f1-score: 0.897) and could even significantly outperform LSA's high recall benchmark of 0.818. Consequently, the main finding of the work presented in this chapter is that *probabilistic features* can convey more valuable information to the classifiers, in order to increase the overall prediction quality. This can be explained by the fact that the probabilistic features reflect statistical correlations between the words which provide more precise descriptions of the actual word-based similarity of the requirement pairs to the classifiers.

Future Work. To counteract common *cold-start problems* which often occur in the early application of a RS, we propose to migrate the existing RS to a hybrid solution which combines the LSA approach with our classification approach based on probabilistic features. Moreover, the evaluation criteria can be relaxed such that, for example, the transitivity of dependencies are considered during the evaluation. A predicted dependency $r_x \rightarrow r_z$ for a given requirement r_x can be considered as correct if there exist two related dependencies $r_x \rightarrow r_y$ and $r_y \rightarrow r_z$ in the test set. Furthermore, our approach can be extended such that further dependency types (e.g., *excludes* or *includes*) can be identified. This can be achieved by treating our classification problem as a multi-class classification problem. However, this would require the use of another (larger) dataset since in the currently used one, only dependencies of type *requires* are included.

Group Recommender User Interfaces for Improving Requirements Prioritization

The contents and results of this chapter are based on the research work published in Samer et al. (2020). The author of this thesis provided major parts of this chapter in terms of the design and evaluation of the user study, writing, and literature research.

5.1. Abstract

Requirements engineering is one of the most critical phases in the context of software development. Unclear textual specifications of requirements, hidden dependencies between requirements, and sub-optimal prioritizations and release plans represent the major reasons for project delays and even cancellation. In this chapter, we show how group recommender user interfaces can help to improve the quality of requirements engineering processes. To that end, we developed a novel group recommendation approach that focuses on the aspect of improving requirements prioritization by making preference elicitation processes more flexible as well as by introducing innovative user interfaces that foster information exchange among stakeholders. We conducted a large user study (N=313 participants) to evaluate our approach. The evaluation results indicate that *argumentation-based user interfaces* in a group setting trigger more rating and communication activity among the group members which significantly improves the quality of the prioritization process. Our main contributions are twofold: (1) more flexibility of the requirements evaluation by supporting the delegation of votes to experts and (2) an increased engagement of the stakeholders responsible for the requirements.

5.2. Introduction

In contrast to single user recommenders (Ricci et al., 2010; Resnick and Varian, 1997), group recommender systems (Masthoff, 2015; Felfernig et al., 2018; Boratto, 2016; Baskin and Krishnamurthi,

2009) focus on scenarios where recommendations are determined for groups of users. The most common way for generating recommendations suitable for a group is to consider the variety of the individual group members' preferences by applying different aggregation strategies (Felfernig et al., 2018; Ninaus, 2012). Nowadays, group recommender systems experience an increasing popularity due to the fact that many recommendation scenarios are group-based. Some examples of such scenarios are the recommendation of holiday destinations (Jameson et al., 2004; Garcia et al., 2009), music recommendations (Crossen et al., 2002; Mezei and Eickhoff, 2017), sequence recommendations (Masthoff, 2004), and resource assignments (Caballero et al., 2014; Felfernig et al., 2018).

In the domain-specific context of *requirements engineering* (RE), there exists a couple of research contributions related to the development and application of recommender systems in RE scenarios (Robillard et al., 2010; Mobasher and Cleland-Huang, 2011; Samer et al., 2019, 2018). These systems address a variety of important application areas inside the RE process such as the detection of unclear textual specifications of software project requirements (Shah and Jinwala, 2015; Berry, 2008) or the detection of dependencies between requirements (Samer et al., 2019; Atas et al., 2018; Carlshamre et al., 2001). In the context of requirements triage, another RE task which is essential for the success of a software project, is the correct prioritization of requirements according to the relevance for the project and the involved stakeholders. An efficient support of prioritization decisions is important as the handling of large assortments of requirements makes manual prioritization processes become very expensive in terms of time and project budget (Xuan et al., 2012). There exist research contributions which aim to tackle this task (Stanik et al., 2018). For example, Alenezi and Banitaan (2013) and Stanik et al. (2018) use machine learning techniques to calculate predictions of requirement priorities. Moreover, Felfernig et al. (2018) present a utility-based prioritization approach for software requirements (issues) in BUGZILLA which exploits relevant meta-data of the requirements in order to determine a user-specific priority of the requirement for the active user / developer.

However, the vast majority of the aforementioned approaches solely focuses on providing recommendation support for single users. Since critical RE-related decisions are usually made by a group of persons (stakeholders), there exists a strong demand for group-based recommendation solutions. To the best of our knowledge, there are group-based approaches which, however, do not go beyond the level of basic manual assessment of the requirements. For example, Duan et al. (2009) introduce an approach to the automated triage of requirements in the context of open-source communities where users are asked to provide estimations individually. Ninaus (2012) (see also Ninaus et al., 2014) present a requirements engineering UI (user interface) environment (INTELLIREQ) that applies preference aggregation functions (Masthoff, 2015) for proposing evaluations of individual requirements acceptable for the whole group. In INTELLIREQ, software requirements are prioritized using group-based multi-attribute utility theory (MAUT), where individual requirements are evaluated with regard to different interest dimensions such as *profit*, *effort*, and *risk*. Based on these evaluations, a priori-

zation can be determined by a utility function.

The major focus of this chapter is to apply group recommendation techniques on the basis of different user interfaces (UI) with the goal to improve the quality of the prioritization of software requirements. The application of group recommender systems is in many cases limited to the aggregation of individual user evaluations of requirements using aggregation functions such as *average* and *least misery* (Masthoff, 2015; Ninaus, 2012; Ninaus et al., 2014). Applying such functions helps to streamline potentially contradictory evaluations. However, such approaches do not take into account the aspects of *liquid democracy* (*delegate voting*) (Johann and Maalej, 2015; Atas et al., 2018), i.e., to make voting processes more flexible and allow to transfer voting rights to stakeholders who are the experts with regard to specific requirements and interest dimensions. In addition to more flexibility on the algorithmic level, user interfaces are required that trigger more stakeholder engagement and help to foster more information exchange between stakeholders which increases the probability of better prioritizations (Atas et al., 2017; Schulz-Hardt et al., 2006). Summarizing, major limitations of existing requirements prioritization approaches are that (1) stakeholder expertise is not taken into account when distributing requirements evaluation tasks and (2) existing user interfaces do not foster information exchange which, however, is a major precondition for assuring high-quality group decisions (Schulz-Hardt et al., 2006). In sharp contrast to existing research contributions, we focus on the development of UI-driven group recommendation approaches that are more deeply integrated into the requirements prioritization process – on the algorithmic level as well as on the level of prioritization user interfaces.

Our major contributions presented in this chapter are the following. First, we propose a new utility-based (argumentation-based) preference elicitation approach that helps to increase information exchange among stakeholders in requirements prioritization. Second, we show how to take into account the concepts of liquid democracy (delegate voting) also to make preference acquisition for requirements prioritization more flexible (on the algorithmic level). Third, we present results of an empirical study that show that the concepts presented in this chapter can improve the quality of requirements prioritization as well as increase the overall evaluation and development engagement of stakeholders involved in a software project. Furthermore, our evaluation results show that our developed approaches had a positive impact on the success rate of the software projects which were implemented within the scope of our empirical user study.

The remainder of this chapter is organized as follows. In Section 5.3, we introduce our approach to liquid-democracy-based requirements prioritization. Thereafter, Section 5.4 presents initial empirical results from a user study which has been conducted within the scope of a university course. Section 5.5 describes the threats both to internal and external validity. A discussion of different issues for future work is given in Section 5.6. Finally, the chapter is concluded with Section 5.7.

5.3. Group Recommendation for Requirements Prioritization

The main objective of our work is to improve the quality of requirements prioritization by taking advantage of a new algorithmic vote delegation concept called *liquid democracy* (Johann and Maalej, 2015; Atas et al., 2018) and a UI-based group recommendation solution which fosters communication among stakeholders. This section presents three different UI variants to determine the priority of software requirements in group-based RE evaluation scenarios. The result of the group recommendation task is represented by a ranked list of requirements based on computed utility values. In general, the utility value of a requirement represents the estimated priority of the requirement based on the users' (stakeholders') evaluations. The used UI variants as well as the vote delegation concept integrated as an extension of these three variants, are discussed in the following.

5.3.1. One-dimensional Rating Approach

One-dimensional ratings represent the conventional (and most convenient) way of evaluating the general importance of a requirement for a software project. Stakeholders are asked to analyse and rate all project's requirements. Based on their assessment, they have to assign a simple rating value (or symbol, e.g., *negative* vs. *positive*) to each requirement. The rating scale is often limited by a small range (e.g., 1 to 5 or 1 to 10) which simplifies and, hence, speeds up the entire evaluation / rating process for stakeholders. Figure 5.1 shows a basic one-dimensional 5-star rating interface to evaluate a given requirement. The utility / priority of the requirement is determined by computing the average (arithmetic mean) of all stakeholder star-ratings provided for the respective requirement. Note that due to the nature of one-dimensional ratings, such ratings often only represent vague and rough estimations of the actual relevance of a requirement for the project in terms of different aspects (such as monetary and technical aspects). However, a varyingly strong consideration of different aspects by stakeholders can often lead to different voting results. For example, a project manager might focus on monetary aspects rather than technical aspects when evaluating a requirement. However, a developer might focus on technical aspects instead.

5.3.2. Multi-attribute Utility Rating Approach

In contrast to one-dimensional ratings (see Section 5.3.1), our approach based on *multi-attribute utility theory* (MAUT) represents a multidimensional evaluation / rating scheme. The basic idea of recommendation methodology based on multi-attribute utility for groups (Ninaus et al., 2014; Felfernig et al., 2018; Atas et al., 2018) is to extend utility-based recommendation for single users to multi-user scenarios where the preferences (evaluations of interest dimensions $d \in D$) of the individual group members are aggregated into a recommendation that is intended to take into account as much as possible the preferences of the whole group. In general, MAUT-based algorithms for groups require from each user to evaluate each alternative with regard to a set of specified interest dimensions. In the context of RE, some examples of such interest dimensions are the potential profit (p) of an implemented

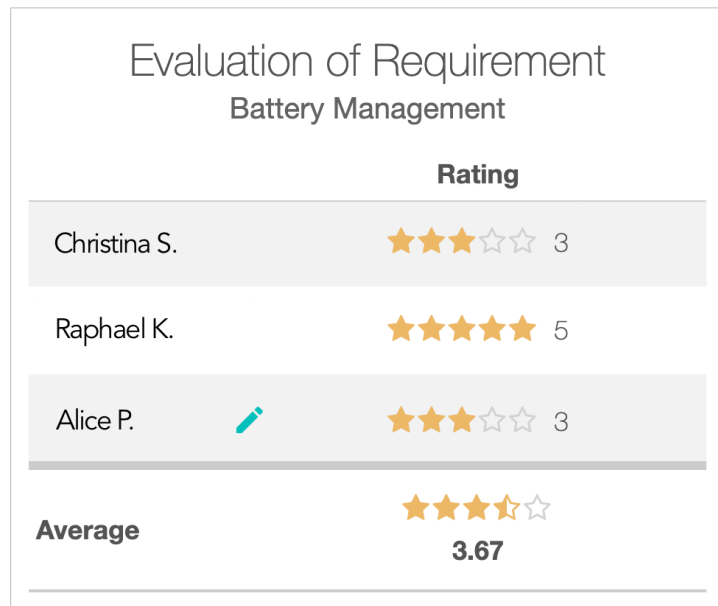


Figure 5.1.: One-dimensional 5-star rating interface. Stakeholders express their estimation of the importance of a requirement using a 5-star rating scale. The average of these votes represents the utility value (here: 3.67).

feature, the effort (e) related to the implementation of a requirement (feature), or the risk (r) of not being able to successfully implement a requirement. Figure 5.2 presents an overview of a basic evaluation example. In this example, three stakeholders evaluated a requirement based on the three interest dimensions *profit*, *effort*, and *risk*. Our empirical evaluation of past research projects with industry partners show that these three interest dimensions represent qualified criteria that were sufficient to evaluate requirements (Ninaus et al., 2014; Atas et al., 2018).

Requirement Priority Estimation. Typically, the evaluations of such interest dimensions provided by the group members¹ are aggregated by using one or multiple (i.e., ensemble) aggregation function(s) (Felfernig et al., 2018; Masthoff, 2015; Ninaus, 2012). This aggregated result is called the calculated *utility value* and represents the fundamental basis on which requirements prioritization takes place (see Figure 5.3). The list of prioritized requirements then further serves as main input for requirements triage as well as for release planning. In contrast to the one-dimensional rating approach where the utility value represents the (aggregated) average of the stakeholders' 5-star ratings, the utility value in our MAUT-based approach is defined as the (aggregated) weighted average of different evaluations / votes given by stakeholders for different interest dimensions. This weighted average utility value directly reflects the priority of a requirement. The aggregation of the different votes (represented by the utility value) constitutes a group decision which benefits from the variety of different

¹In the context of RE, the group members are stakeholders.

opinions (Schulz-Hardt et al., 2006). A high utility value indicates a high relevance / priority of the requirement, whereas a low utility value signifies a low relevance of the requirement for the project. The utility values of the requirements directly influence critical decisions in the release planning phase of a software project – especially, in which releases the requirements should be implemented (*which requirements should be assigned to earlier and which requirements should be assigned to later releases*). Thus, an accurate estimation of the utility values plays a crucial role in the release planning phase of a software project. This has a large impact on the success of the software project, because the late discovery of wrong assignments (of requirements to releases) can lead to negative consequences, such as increased costs, unfulfilled deadlines, or even project failure. Formula 5.1 shows the utility calculation of requirement r based on the estimations of the stakeholders who evaluated r . In this formula, $imp(d)$ denotes the estimated importance / weight of the interest dimension d . The expression $eval(r, s, d)$ represents the rating value chosen by stakeholder s for the interest dimension d of requirement r . Finally, $w(s, d)$ reflects the expertise level of stakeholder s with respect to interest dimension d and the total number of stakeholders is represented by $|S|$.

Evaluation of Requirement Battery Management			
	💰 Profit	⚠ Risk	🔧 Effort
Elizabeth R.	7	5	6
William S.	7	5	5
Adelaide L.	9	4	7
Average	7.67	4.67	6.00

Figure 5.2.: Overview of multidimensional MAUT ratings for a requirement. In this example, three stakeholders evaluated a requirement based on the interest dimensions *profit*, *risk*, and *effort*. Given these votes, the utility value can be calculated using Formula 5.1.

$$util(r) = \frac{\sum_{s \in S} \frac{\sum_{d \in D} eval(r, s, d) * imp(d) * w(s, d)}{\sum_{d \in D} imp(d) * w(s, d)}}{|S|} \quad (5.1)$$

ID	Title	Description	Status	Utility
R44	PDF Generation	Export project information to pdf (https://mast-tuleap.informatik.uni-hamburg.de/plugins/tracker)	New	5.25 5 user voted
R48	Requirement Attachments	Add and modify attachments of a requirement; Users can only upload (i.e., add) attachments and	New	3.80 4 users voted
R27	Requirement Recommendation	Extraction out of documents	New	2.75 4 users voted

Figure 5.3.: Recommended prioritization of requirements. The utility values (see right side) reflect the evaluated priority of a requirement and are calculated based on the evaluations provided by the stakeholders for the specific requirement. A high utility value of a requirement indicates a high priority which advises decision makers (e.g., requirements managers) to consider the requirement in earlier releases rather than requirements of a lower utility.

Liquid Democracy Extension. Since different persons involved in a project usually have different background knowledge and opinions, requirements should be evaluated by different stakeholders (i.e., customers, developers, project managers, etc.) individually as well as independently. In terms of liquid democracy, stakeholders can either evaluate the interest dimensions directly or delegate their vote for a specific interest dimension (or requirement) to a stakeholder who is more qualified to evaluate this dimension / requirement. Each stakeholder $s \in S$ can either decide to (1) evaluate the interest dimension (or requirement) directly or to (2) delegate the vote of an interest dimension (or requirement) to a different stakeholder $s' \in S \setminus \{s\}$. In case of liquid democracy, the expression $eval(r, s, d)$ presented in Formula 5.1 denotes either the estimate of stakeholder s directly or the estimate of a stakeholder $s' \neq s$ to whom stakeholder s delegated the vote (i.e., the estimation task). The function $w(s, d)$ represents the weight of the stakeholder s reflecting the general expertise (knowledge / skill level) of either s (if s voted) or s' (if s delegated the vote to s') for dimension d . Since every stakeholder can delegate the vote for an interest dimension to only one other stakeholder, the complete vote-inheritance graph of delegated votes of a stakeholder for one interest dimension can be visualized in a (directed) tree.

Figure 5.4 shows an example of such a delegation hierarchy for one interest dimension². Only the stakeholder who represents the root node in a delegation tree (stakeholder s_1) votes for the interest dimension. The vote of this stakeholder is then inherited to all other stakeholders (here: s_2 - s_6) who appear in the same tree. In order to avoid cycles in the graph and to ensure that the graph always

²A delegation hierarchy for delegating the vote of a *complete* requirement instead of a single interest dimensions would look identical to the example described in Figure 5.4.

remains a tree structure, our approach does not allow back-delegation of the votes within the tree. For instance, if the stakeholders s_5 and s_6 would delegate their vote to stakeholder s_4 and stakeholder s_4 would further delegate the vote to stakeholder s_2 , stakeholder s_2 would not be allowed to delegate the vote back to stakeholder s_4 , s_5 , or s_6 any more. This cycle-protection mechanism utilizes the *depth-first search* (Tarjan, 1971) algorithmic approach and is triggered before a stakeholder delegates the vote to another person.

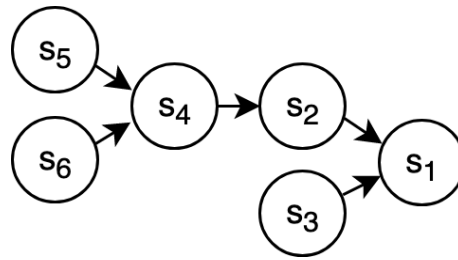


Figure 5.4.: Basic example of a vote-inheritance tree for one interest dimension (or one requirement). In terms of liquid democracy, stakeholders (nodes) can delegate their vote to a different stakeholder (directed edge).

Table 5.1 depicts a basic evaluation scenario where individual stakeholders evaluate the requirements $\{r_1, r_2, r_3\}$ with regard to the interest dimensions *profit*, *effort*, and *risk*. The values in brackets (Tables 5.1 and 5.2) indicate situations where an evaluation for an interest dimension has been delegated. In our example, *Ann* is not an expert in evaluating the potential *profit* of a requirement and delegated this evaluation task to *Chris*. *Chris* and *Susan* did not transfer any votes. This simplified example leads to a situation where *Ann* and *Chris* provide the same evaluations for the dimension *profit*. Often, votes are transferred on a requirement-specific but not an interest dimension-specific level. Furthermore, we assume that stakeholder expertise is defined on the level of interest dimensions (see Table 5.2). Note that these values can either be specified by the stakeholders directly, automatically derived from previous requirements evaluation activities³, or inherited via liquid democracy (see *Ann* in our example). The utility value of r_1, r_2, r_3 can be determined by applying Formula 5.1 to the votes presented in Table 5.1 and by using the importance weights $imp("profit")=0.5$, $imp("effort")=0.4$, $imp("risk")=0.3$ and the (expertise) weights shown in Table 5.2. A high utility value indicates a high priority and hence a low rank in the ordered list of prioritized requirements (see last row of Table 5.1). The prioritization can be seen as a recommendation for a sequence in which r_1, r_2, r_3 should be implemented (in our example, the recommended sequence is $[r_2, r_3, r_1]$).

³The automated estimation of stakeholder expertise is an issue for future research.

Stakeholder	r_1			r_2			r_3		
	p	e	r	p	e	r	p	e	r
Ann	(3)	5	2	(5)	5	5	(2)	8	8
Chris	3	8	4	5	4	6	2	6	7
Susan	5	3	5	7	5	9	1	7	6
Utility value (priority)	4.34			5.68			4.86		
Priority ranking	3			1			2		

Table 5.1.: Three stakeholders evaluated the requirements $\{r_1, r_2, r_3\}$ with regard to the dimensions profit (p), effort (e), risk (r). The calculation of the utility values is based on Formula 5.1. The priority ranking defines the suggested ranking of the requirements based on their determined utility.

Stakeholder	profit(p)	effort(e)	risk(r)
Ann	(0.3)	0.7	0.2
Chris	0.3	0.2	0.3
Susan	0.4	0.3	0.4

Table 5.2.: Assumed expertise of the stakeholders $\{Ann, Chris, Susan\}$ in the range between 0.0 and 1.0 with regard to the dimensions profit (p), effort (e), and risk (r) (0.0 = very low ... 1.0 = very high).

5.3.3. Argumentation-based Rating Approach

In order to increase the flexibility of the requirements evaluation, we also focused on fostering communication among stakeholders (Schulz-Hardt et al., 2006; Al-Rawas and Easterbrook, 1996; Coughlan and Macredie, 2002). For this purpose, we extended the MAUT-based evaluation approach which was introduced in Section 5.3.2. Our argumentation-based rating approach is based on the MAUT-based approach and uses an argumentation-based user interface for eliciting dimension-specific evaluations. The underlying idea of our approach is that a higher degree of information exchange between persons involved in a group decision, has a positive impact on the quality of the group decision (see also Greitemeyer and Schulz-Hardt, 2003 and Mojzisch and Schulz-Hardt, 2010). The rating interface of the argumentation-based approach allows stakeholders to provide descriptive arguments (in terms of comments) against a specific requirement or arguments which support a specific requirement. An example of such an argumentation-based user interface is depicted in Figure 5.5. In order to distinguish between positive and negative arguments, stakeholders who enter an argument have to manually classify the sentiment of the argument. Thereby, the stakeholder is asked to mark his / her argument as either *positive* (PRO), *neutral* (NEU), or *negative* (CON).⁴ Moreover, the stakeholder also has to assign an interest dimension (e.g., *profit*, *effort*, *risk*) to the argument. Every textually-defined argument can be interpreted as a requirement rating / vote of one specific interest dimension with a sentiment type (i.e., PRO, CON, NEU) selected by the stakeholder.

⁴Future versions of our rating interface may provide support to the stakeholders with an automated sentiment classification of the arguments based on machine learning.

In terms of MAUT, a fixed rating value can be used based on the argument’s sentiment type to calculate a utility value for the requirement. For example, when using the value-scheme {PRO=3, NEU=2, CON=1}, the positive PRO-argument "Nice feature" for the requirement "PDF Generation" shown in Figure 5.5 would represent a MAUT-vote of 3 points dedicated to the interest dimension *profit*. This way, a utility value for each requirement can be calculated in the same fashion as presented in Section 5.3.2 by applying Formula 5.1.

Currently, arguments are primarily regarded as a means to foster communication among stakeholders which is regarded as one of the major means to increase the quality of decisions in social psychology research. In order to be able to make high-quality decisions, it is important that all group members have the decision-relevant information available (for related details we refer to Schulz-Hardt et al., 2006).

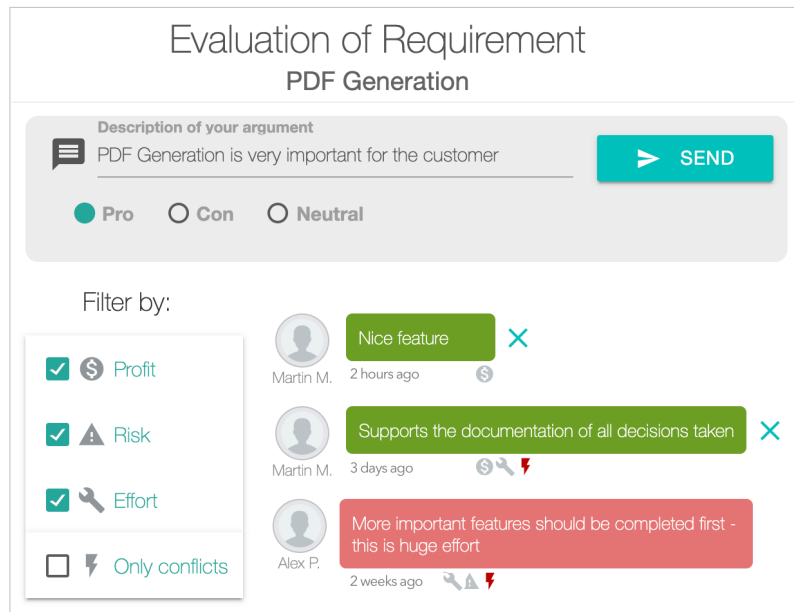


Figure 5.5.: Argumentation-based rating interface. In addition to the basic MAUT version, stakeholders are enabled to exchange arguments for / against specific requirements. Positive arguments are highlighted in green, neutral arguments in orange, and negative arguments in red. After entering the text for an argument, the system asks for the interest dimension that touches the user’s argument.

5.4. Evaluation

In this section we report the results of our empirical evaluation. N=313 computer science students participated in a course on object-oriented analysis and design. Their task was to develop a tourist

information software including a recommendation functionality for tourist destinations (the 313 students worked in development teams consisting of 4–6 students, 60 groups in total). For the purpose of requirements prioritization, the students had to maintain their software project and apply our prioritization functionality provided by a prototype system called OPENREQ!LIVE⁵ which has been developed within the scope of the research project OPENREQ funded by the European Union. The students worked in groups and defined, evaluated, and prioritized the requirements for the implementation of the tourist information platform by using OPENREQ!LIVE directly.

5.4.1. Experimental Setup

The major focus of our empirical user study was to analyze the impact of the three different preference elicitation user interfaces (presented in Section 5.3) on the quality of the final prioritization. For the purpose of comparing the UI variants, we assigned different variants to the development teams (between-subjects design). First, a *rating scale based version* supported the rating of requirements on the basis of a one-dimensional 5-star rating scale. In this version, the group members had to provide a *single* rating (1-5 stars) for every requirement of their project (see Section 5.3.1 and Figure 5.1). Second, a *group-based multi-attribute utility* (MAUT) based approach was used to determine a prioritization. Here, the group members had to evaluate (1-5 points) the interest dimensions *profit*, *effort*, and *risk* individually for every requirement of their project (see Section 5.3.2 and Figure 5.2). Third, again a MAUT-based interface was provided. In contrast to the basic group-based MAUT approach, this variant included an *argumentation-based interface* (see Section 5.3.3 and Figure 5.5). The students were asked to comment on issues for every requirement. Every requirement was discussed individually by the group (i.e., a separate discussion thread existed for each requirement). In such a discussion thread, the group members could provide arguments and mark them as either *positive* (PRO), *neutral* (NEU), or *negative* (CON) based on the sentiment of the argument. In addition to that, every given argument had to be assigned to one interest dimension (i.e., *profit*, *effort*, or *risk*).

In the context of this user study, 20 teams (105 students) were assigned to the 5-star based rating scale user interface, 20 teams (104 students) to the basic MAUT version, and 20 teams (104 students) to the argumentation-based MAUT user interface. We equally distributed groups with different sizes (4–6 members) to the three UI variants. One team member was selected by the group to be responsible for the administration of the group’s software project (i.e., stakeholder role: project manager). All other team members participated in the project as developers (i.e., stakeholder role: developer). In order to counteract *cognitive biases* (Tversky and Kahneman, 1975), the groups were not informed about the existence of different UI variants during the study. Especially, to avoid *anchoring effects* (Mojzisch and Schulz-Hardt, 2010; Schulz-Hardt et al., 2006; Tversky and Kahneman, 1975), the ratings which have already been provided by the other team members as well as the current average and the utility values of a requirement were not shown before a student evaluated / rated the requirement. In all three

⁵OPENREQ!LIVE: <https://github.com/OpenReqEU/openreq-live>

UI variants, the utility value defined the basis for the groups to prioritize the requirements and to do their release planning ("which requirements are assigned to earlier / later releases"). In the 5-star based rating version, the utility value of a requirement represented the average of the 5-star ratings of this requirement. The utility values used in the basic MAUT and argumentation-based MAUT versions were calculated based on Formula 5.1. To ensure comparability of the utility values across all versions, we used the same rating scale (from 1 to 5 points / stars) in all three versions. For all interest dimensions and all students / stakeholders, we set $w(s, d) = 1$ (see Formula 5.1) at the beginning. Based on the individual voting-delegation behavior (*liquid democracy*), $w(s, d)$ was higher than 1 for stakeholders who received delegated voting power from other students. In the argumentation-based MAUT version there were no *direct* ratings but the students had to select one of three sentiment levels (PRO, NEU, CON) for each of their arguments (corresponding to one interest dimension). Thus, in order to compute a utility value for this version, we assigned 5 points to every PRO-argument, 3 points to every NEU-argument, and 1 point to every CON-argument (see also Section 5.3.3). We used this point scheme in the argumentation-based MAUT version because we wanted to use the same point range (from 1 to 5, where 1 represents the lowest possible (CON) and 5 the highest possible rating value (PRO)) in all three UI variants. Therefore, we mapped the whole point range to these three sentiment levels. In contrast to the other two UI variants, we limited the argumentation-based rating interface to only these three sentiment levels in order to allow the users to focus on writing more qualitative arguments instead of spending too much time on giving fine-grained point-ratings (which would include ratings of 2 and 4 points).

To assess the quality of the final requirements prioritization determined by the groups (based on the computed utility value of the requirements), we decided to evaluate the quality of the prioritization in terms of the overall outcome of the software development process. In our case, the software components developed by the students were evaluated with regard to criteria such as *software quality*, *reusability*, and *completeness* in terms of the coverage of the planned requirements. The final assessment and grading of the software components developed by the groups with respect to the aforementioned criteria was performed by four study assistants who were not informed about which of the three UI variants was assigned to the groups. We randomly assigned 15 groups for assessment to each study assistant by considering an equal distribution among the three UI variants (i.e., every study assistant had to assess 5 groups of *each* UI variant). This counteracts behavioral differences of the study assistants in the assessment and grading process which helps to draw more solid conclusions. Moreover, we also measured detailed statistics of the user activity and evaluated the results to look for significant differences between the UI versions.

5.4.2. Results & Discussion

Table 5.3 provides a general overview of major observations from our study results showing the number of requirements, the number of evaluations, and the total number of rating / evaluation interactions

for all three UI versions (5-star rating, group-based MAUT, and argumentation-based MAUT). Figure 5.6 presents a visual comparison of statistical details regarding the measured information of all three UI variants. The figure visualizes the distributions of (a) ratings per requirement, (b) rating adaptations / interactions per requirement (this includes create, update, and delete of evaluations)⁶, and (c) achieved points by the students at the end of the user study. All corresponding numeric results of these plots can be found in the Appendix of this thesis (see Table A.1 in Appendix A).

As can be seen in Table 5.3, the groups working with the 5-star version defined more requirements per group on average (531 requirements / 20 groups = 26.55) compared to the other versions (group-based MAUT: 22.95, argumentation-based MAUT: 16.50). Groups that were confronted with the argumentation-based rating system showed the highest number of average evaluations per requirement - they provided 5.1 arguments / evaluations for a requirement on average (see Figure 5.6a), the groups which used the basic MAUT version created only 4.2 ratings / evaluations per requirement on average, whereas the groups of the 5-star rating system provided only 3.9 evaluations per requirement on average. In order to underpin our findings, we evaluated all three versions for statistical significance using two-sample *t-tests* ($\alpha = 0.05$). Regarding the number of ratings / evaluations per requirement, our evaluation confirms that groups which used argumentation-based rating systems have evaluated requirements significantly more frequently than groups using the 5-star (t-test: $t = -17.89$, $p < 0.01$) or basic MAUT version (t-test: $t = -12.76$, $p < 0.01$). A comparison between the 5-star rating and the basic MAUT version reveals that groups using the basic MAUT version have provided significantly more ratings (t-test: $t = -5.18$, $p < 0.01$) than the other groups.

Furthermore, the argumentation-based MAUT version showed the highest interaction rate (3433 interactions / 330 requirements = 10.4 interactions per requirement on average, standard deviation: 1.03; see Figure 5.6b). In this context, the interaction rate represents the number of rating adaptations. The interaction rates of the basic MAUT version (average: 5.4, standard deviation: 0.65) and the 5-star rating UI version (avg.: 4.5, std. dev.: 0.61) were significantly lower compared to the argumentation-based version (5-star version: $t = -94.13$, $p < 0.01$; basic MAUT version: $t = -77.32$, $p < 0.01$). A statistical significance in terms of more rating interactions can also be observed in the basic MAUT version when compared to the 5-star rating version ($t = -23.06$, $p < 0.01$).

This strong tendency of more rating adaptations (rating interactions) in the argumentation-based MAUT version can be explained by a slow (but more qualitative) convergence of the decision process due to a higher probability of having available a more decision-relevant knowledge (in terms of more qualitative arguments) for the prioritization. Group-based MAUT without argumentation support also has a higher amount of average evaluations per requirement which can be explained by the fact that a utility-based preference elicitation fosters a more detailed analysis of requirements along different prioritization-relevant dimensions (Stettinger et al., 2015). Additionally, due to the existence

⁶However, the number of updated evaluations / arguments is 0 for argumentation-based evaluations because the system only allows its users to create and delete arguments.

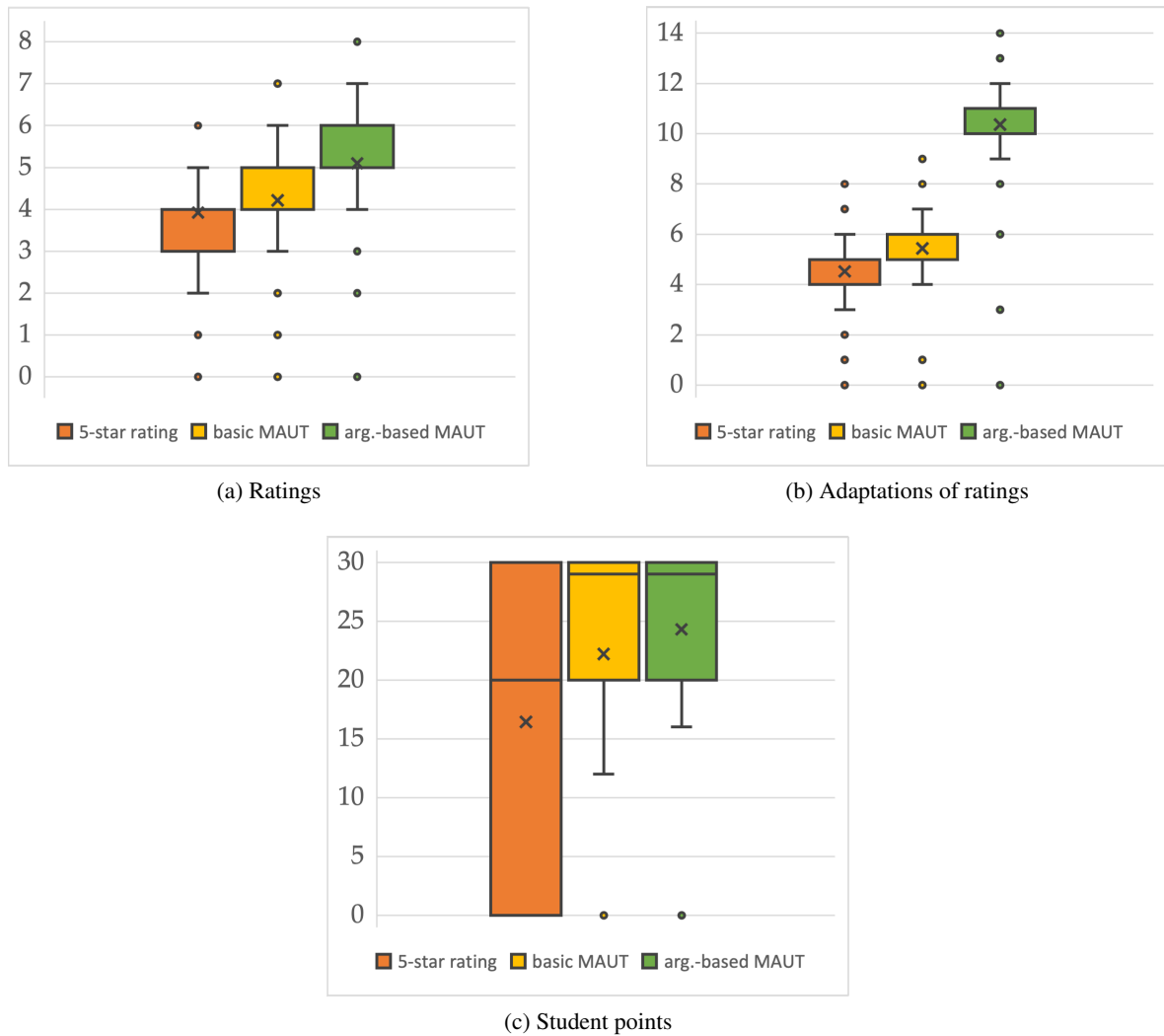


Figure 5.6.: Study results of our three UI types showing statistical details about the number of (a) *rating evaluations*, (b) *rating adaptations / interactions*, and (c) *points achieved by the students* (maximum number of points = 30). In all three plots, the marker "x" represents the mean of the distribution and all outliers are shown in the form of the symbol "o". The median values of plots (a) and (b) cannot be identified since they either overlap with the first or the third quartile. The median values of these plots are as follows: (a) 5-star rating: 4, basic MAUT: 4, argumentation-based MAUT: 5; (b) 5-star rating: 5, basic MAUT: 5, argumentation-based MAUT: 10.

UI type	requirements	evaluations	rating interactions
5-star rating	531	2,084	2,393
basic MAUT	459	1,935	2,493
argumentation-based MAUT	330	1,683	3,433

Table 5.3.: Overview of our study results showing general details about the number of requirements, the number of evaluations, and the total number of rating / evaluation interactions with regard to the corresponding UI version (5-star rating, basic MAUT, argumentation-based MAUT).

of multiple dimensions in group-based MAUT which reflect different evaluation aspects (e.g., profit vs. effort), also more oral communication might be triggered within the group. In contrast to that, 5-star based rating scales do not trigger an analysis along requirement-relevant dimensions and also lead to a focus on achieving fast consensus (i.e., low interaction rate) without deeply taking into account decision-relevant knowledge. This results in situations where groups seem to have achieved fast consensus without a detailed analysis of the requirements, i.e., fast convergence of the requirements prioritization process at the cost of a limited quality. Moreover, the phenomenon that significantly more requirements per group were defined on average in the 5-star version can also be explained by the reason that this basic rating system does not trigger a detailed analysis of the requirements as already mentioned before. This often leads to too optimistic and unrealistic estimations of the requirement's effort and relevance (Catania, 2006; Maguire and Bevan, 2002). The serious consequences of such insufficient estimations (caused by 5-star rating scales) can be the implementation of useless features, exceeding costs, or even project failure (Greer and Ruhe, 2004; Ruhe, 2010).

In order to evaluate the impact of the users' rating tendency, we analyzed the sentiment of the ratings. All three versions were evaluated for significance (one-sample t-test, $\alpha = 0.05$). We tested each version individually whether the ratings (ranging between 1 and 5 stars / points) tend towards a positive ($H_0 : \mu \leq 3$) or a negative ($H_0 : \mu \geq 3$) rating sentiment. Table 5.4 shows the number of evaluations which were positive, negative, or none of both (neutral) for all three versions. In the basic MAUT version, the presented numbers include all ratings of the three interest-dimensions as separate ratings (1,935 requirement ratings * 3 dimensions = 5,805 total evaluations). For the purpose of comparing our three different UI variants in terms of the sentiment, we defined an "associative point scheme". We considered ratings with 1 or 2 stars / points as negative ratings, ratings with 3 points as neutral ratings, and ratings with more than 3 points as positive ratings. In our 5-star rating version, the overall sentiment of the ratings (46.35% positive ratings) was positive (t-test: $t = 13.55$, $p < 0.01$). The overall sentiment of the basic MAUT evaluations (43.84% negative ratings) was negative (t-test: $t = -16.11$, $p < 0.01$). Likewise, the overall sentiment of arguments in the argumentation-based MAUT version tended to be negative as well (t-test: $t = -13.27$, $p < 0.01$).

This implies that both MAUT-based versions seem to foster a more critical analysis and evaluation of the requirements (negative sentiment of the ratings dominates) compared to 5-star ratings (positive sentiment of the ratings dominates). This can be interpreted as a more conflict-oriented assessment of the requirements that triggers more discussions and reevaluations of the requirements (adaptations of evaluations). In fact, as previously mentioned, both MAUT-based versions achieved a significantly higher rating interaction / adaptation rate than the one-dimensional 5-star version.

UI type	Negative (1-2)		Neutral (3)		Positive (4-5)	
	absolute	in %	absolute	in %	absolute	in %
5-star rating	473	22.69	645	30.95	966	46.35
basic MAUT	2545	43.84	1721	29.65	1539	26.51
arg.-based MAUT	1005	59.71	165	9.80	513	30.48

Table 5.4.: Statistical sentiment analysis of the requirement evaluations. Based on the rating value (ranging from 1 to 5) we classified the requirement ratings / evaluations into three different sentiment levels (negative: 1 and 2, neutral: 3, positive: 4 and 5). The dominating sentiment levels for each UI type are highlighted in bold.

In order to analyze potential impacts of the three user interfaces on the output quality of the software development process, we also compared the points the group members received for their developed software components at the end of the course (maximum number of points = 30). As mentioned, the developed software products have been evaluated with regard to certain criteria such as *software quality*, *reusability*, and *completeness* in terms of the share of originally scheduled (within the scope of the prioritization process) and successfully implemented requirements. Figure 5.6c depicts the distribution of the points the students achieved for their developed software components. The best results (on average) were achieved by students who used the argumentation-based MAUT version (24.31 points, basic MAUT: 22.20 points, 5-star: 16.44 points). However, the data shows high standard deviations of the points in case of all three versions (see Figure 5.6c and Table A.1). Hence, we conducted two-sample t-tests ($\alpha = 0.05$) in order to check statistical significance of this data pairwise between the different versions (i.e., $H_0 : \mu_x = \mu_y$). Our results indicate that there exists a significant difference in the data between the 5-star rating scale and the basic MAUT user interface (t-test: $t = -3.37$, $p < 0.01$). The same holds true for the comparison of the 5-star rating version with the argumentation-based MAUT version (t-test: $t = -5.12$, $p < 0.01$). However, a difference between argumentation-based MAUT and standard MAUT can only be observed on a descriptive level (t-test: $t = -1.55$, $p = 0.06$, $p > \alpha$).

Based on the results presented in this section, we conclude that an argumentation-based MAUT UI can improve the quality of the prioritization process by significantly increasing the number of rating adaptations. The (adapted) arguments also represent a better basis for the prioritization of the requirements and for the release planning which is essential for project success. Moreover, we see one reasonable conclusion as follows. Due to the increased rating interaction triggered by the argumentation-based

requirement evaluation, the quality of the resulting software projects rises as indicated by the final points achieved by the students. More numeric results and results of significance tests, which further confirm our findings, can be found in the appendix of this thesis (see Appendix A).

5.5. Threats to Validity

In this section, we shed light on threats to the internal and external validity of our presented group recommender user interfaces for requirements prioritization. These threats are briefly explained and discussed in the following.

5.5.1. Internal Validity

One potential internal threat is that our prototype has only been evaluated with computer science students. In each group, one student was responsible for the administration of the group's project (project manager). The role of a developer was assigned to all other team members and hence represented the dominating stakeholder role in the study. Moreover, we used a rating scale ranging from 1 to 5 points / stars in all three UI versions in our empirical study. A different more fine-grained rating scale (e.g., from 1 to 10) might lead to (minor) deviations of the stakeholders' rating behavior (Gena et al., 2011). Furthermore, the size of development teams (groups) in our study was limited to 4–6 team members. Although the evaluated results of the presented work already represent a solid foundation to build upon (due to the large number of study participants (N=313) and the proven significance of the measured data), the results may differ with larger varying group sizes (e.g., hundreds of stakeholders). In the rare case of a few groups (less than 3%), we observed that not all team members evaluated all requirements. Even though this might have slightly biased our data, this effect can be safely ignored due to its small impact on the results and the fact that not all of these groups used the same UI version for requirements prioritization.

5.5.2. External Validity

The explanatory power of the evaluation results is limited since the evaluation of our requirements prioritization UI solutions was solely conducted within the scope of one specific domain (i.e., the development of a tourist information system). A reevaluation of our approaches for different domains may lead to (small) deviations of our results. Moreover, an industrial operation of our system could also lead to minor deviations of our results because the presented results originate from a controlled experimental setting. Another threat to external validity is that we focused our evaluation on a university setting which provided the possibility of comparing different variants of a group-based prioritization interface. Finally, further research must prove the robustness of our approach against various rating-behavioral peculiarities caused by the visual representation / appearance of the user interfaces.

5.6. Future Work

There are a couple of open issues for future research. Within the scope of future work we will address the major issues of internal and external validity (see Section 5.5). In particular, we plan to evaluate our recommendation approaches in the context of different domains in order to confirm our evaluation results. In order to obtain more representative results and to further prove the significance of our results, our approach will be evaluated with a variety of different stakeholder roles (i.e., customers from various branches, requirements engineers, project managers, and developers). Moreover, we will compare our approaches with further well-known requirements prioritization approaches to prove the potential of our approaches. Additionally, we are currently conducting user studies that focus on the evaluation of the usability of the different user interfaces. Initial results from (early) usability tests (and a user questionnaire) with industrial partners indicate that argumentation-based interfaces are highly appreciated by the community. Therefore, another task for future work is to prepare the existing prototype to conduct studies in real-world industrial settings. Furthermore, we plan to integrate functionalities that support the recommendation of project teams. An important issue in this context is to not only provide a recommendation of a group configuration based on expertise criteria but to also consider availability aspects of individual stakeholders. Finally, we will conduct a semantic analysis on how liquid democracy influences the arguments given in the discussion and how the competence of the participants can be inferred from the existing rating behavior based on specific aspects and the flow of control.

5.7. Conclusions

In this chapter, we introduced two concepts that both can help to improve the decision quality in the context of requirements prioritization processes. Our first contribution are liquid democracy-based extensions which represent a novel concept for multi-attribute utility evaluations that allows stakeholders to delegate votes to other stakeholders who have more expertise on particular aspects (dimensions) of requirements. It is a simple and elegant solution for requirement assessment that helps to capture more and better-quality information in calculating the utility of different multi-aspect requirement-related options which is essential in recommending a proper prioritization of requirements. Our second contribution of this work represents a novel argumentation-based user interface of a chat-like forum that allows the elicitation of requirement preferences of users and engages them to argue for or against certain requirement dimensions. Our findings show that such argumentation-based user interfaces lead to an increased communication and interaction rate in terms of more exchanged as well as more frequently updated (textual) arguments. This way of completing a prioritization task serves as a basis for high-quality prioritizations.

Group Decision Support for Requirements Management Processes

This chapter is based on the contents published in Samer et al. (2018). The author of this thesis wrote most parts of this chapter and provided major contributions in terms of literature research and the design of the presented group recommendation approach. For the innovative ideas presented in this work and chapter, we received the SIEMENS Student Best Paper Award at the 20th International Workshop on Configuration in 2018 (ConfWS 2018).

6.1. Abstract

Requests for proposal (RFP) trigger company-internal requirements management (RM) processes in order to assure that offers comply with a given set of customer requirements. As traditional RM approaches require a deep involvement of the requirements managers of a RM project especially when it comes to assigning suitable stakeholders to requirements, the quality of the decisions and the time effort for making correct decisions mainly depends on these experts. In this chapter, we present a novel stakeholder assignment approach that reduces the overall involvement of these experts and also limits the uncertainty of overseeing suitable stakeholders at the same time. The assignment of responsible stakeholders is represented as a *group decision task* expressed in the form of a basic configuration problem. The outcome of such a task is a configuration which is represented in terms of an assignment of responsible stakeholders to corresponding requirements.

6.2. Introduction

Group-based configuration is an important application area of artificial intelligence (Felfernig et al., 2016, 2014). It aims to support a group of users in the configuration of complex products or services. In general, when interacting with group-based configurators, group members first articulate

their preferences, then adapt inconsistent constraints, and finally, solutions are generated (i.e., reflecting the given configuration). In particular, when interacting with a configurator in the context of a typical requirements engineering task, each group member (i.e., stakeholder) has to evaluate each requirement according to different dimensions such as *priority*, *effort*, and *taken risk*. However, for the definition and evaluation of these requirements, first, suitable stakeholders have to be identified who are responsible for the development of these requirements. In addition, an early involvement of these stakeholders in the project is essential for the success of a project (Felfernig et al., 2017; Hofmann and Lehner, 2001; Leffingwell, 1997; Mobasher and Cleland-Huang, 2011). This is because a low involvement of stakeholders in a project can lead to project failure. Project failures are often caused by missing or wrong assignments of stakeholders to requirements in early phases of the *requirements engineering process* (Lim et al., 2010). Stakeholder recommendations can help to identify persons who are capable of providing a complete analysis and description of software requirements. Recommended stakeholders also need to bring deep knowledge about the corresponding item domain in order to provide precise evaluations of the requirements.

STAKENET (Lim et al., 2010) is an application that supports stakeholder identification on the basis of *social network analysis*. This approach builds a social network on the basis of a set of stakeholders. In this social network, stakeholders are represented as *nodes* and recommendations articulated by the stakeholders are represented as *links*. On the basis of such social networks, different *social network measures* are used for the prioritization of the stakeholders. One example of such a measure is *betweenness centrality* which measures the priority of a certain stakeholder *s* based on the ability of this user to play a role as a *broker* between separate groups of stakeholders. Castro-Herrera et al. (2008) and Mobasher and Cleland-Huang (2011) introduce a content-based recommendation approach where requirements are grouped by using different clustering techniques. Subsequently, stakeholders are recommended and assigned to these groups on the basis of content-based filtering.

In this chapter, a novel stakeholder assignment approach is introduced. The presented approach, acting as basic configuration service, lets voters evaluate stakeholders based on different criteria / dimensions and then aggregates their votes to derive possible configurations which are then recommended for the final stakeholder assignment decision to the requirements manager. In contrast to the aforementioned stakeholder recommendation approaches where the generated recommendations are directly suggested to the requirements manager, the *content-based recommendation service* presented in this chapter *only* acts as a single *artificial voter* in addition to some human voters. Hence, the stakeholder recommendations (i.e., possible configurations) shown to the requirements manager are determined based on a combination of votes reflecting opinions of human voters as well as votes reflecting opinions of artificial voters.

The major contributions of this chapter are the following. First, we analyze in detail a real-world

scenario of a typical bid project. Second, we show an approach to identify relevant stakeholders for specific requirements and thus generate a global assignment of stakeholders to requirements. The remainder of this chapter is organized as follows. In Section 6.3, we describe a typical application scenario of a bid project applied in an industrial context and provide a practical view of a traditional requirements management process commonly used for planning large industry projects. Additionally, a novel sophisticated approach is explained which further improves and extends the traditional approach by considering group decision support techniques. Section 6.4 discusses some potential issues and several factors this approach depends on. Subsequently, Section 6.5 explains the implementation of such an approach from a technical viewpoint. Finally, Section 6.6 concludes with a brief recap of this chapter and presents some ideas for future work.

6.3. Application Scenario

Whenever an organizational unit of a large company (e.g., SIEMENS¹) decides to bid for a *request for proposal* (RFP), a new bid project for that proposal is initiated and the necessary stakeholders of the bid project are identified. RFPs for technical systems usually consist of a set of PDF (*Portable Document Format*) or Microsoft Word documents which describe all requirements for the requested system covering technical, financial, or legal aspects. Examples of stakeholders can be project managers, system architects, requirements managers, quality management departments, legal departments, engineering departments relevant for the bid, and potential external suppliers.

Within the context of a bid project, a requirements management (RM) process is initiated at the beginning. The purpose of this process is to assure that no requirement of the RFP has been overlooked. It involves the extraction of all the requirements contained in the RFP documents. The identified requirements must be assessed by the relevant stakeholders. This means that requirements concerning contracts must be assessed by the stakeholder(s) of the legal department, technical requirements must be assessed by the affected engineering department, etc. The assessment may involve statements about various criteria such as compliance, risks, or approaches. These statements are interpreted as *evaluation dimensions* in the remainder of this chapter. At the end, each requirement of the RFP must have been assessed by at least one appropriate stakeholder.

6.3.1. Traditional RM Process

The *traditional requirements management* process can be best explained with an example. In the following, we describe a simplified example of a traditional RM process in a rail automation context based on a conventional RM tool such as IBM DOORS².

¹SIEMENS: <https://www.siemens.com>

²IBM DOORS: <https://www.ibm.com/products/requirements-management>

Domain	Stakeholder
PM	project manager
SA	system architect
RM	requirements manager
RAMS	reliability, availability, maintainability, and safety
S(ignal)	engineering department for railway signals
PS	engineering department for power supply
TVD	department for track vacancy detection
ETCS	department for European Train Control System
Test	quality management department
Supplier1	external supplier, subcontractor

Table 6.1.: Examples of domain identifiers for rail automation

At the beginning, the requirements manager of the bid project creates a new project in the RM tool. After that, the necessary stakeholders for the current bid project are defined. In this context, stakeholders do not necessarily correspond to persons but correspond to roles which are uniquely identified with a unique string (called *domain*). These string-based identifiers are unique within the organization. Furthermore, the RM tool supports the mapping of existing roles (i.e., domain identifiers) to concrete persons within the bid project. This way, responsible persons are assigned to roles based on their skills and domain knowledge.

Table 6.1 presents some examples of domain identifiers which occur in the context of rail automation. For such large bid projects usually more than 50 different domains are defined with the RM tool. However, in practice, most projects only use 20 different domains on average.

As a next step, the requirements manager imports all the relevant documents of the RFP into the project by using the RM tool. The RM tool automatically converts each paragraph of the documents into a (potential) requirement whilst the structure of the documents is preserved. The requirements manager then classifies the (potential) requirements in the project as either an actual requirement or as an arbitrary comment (called prose). In general, large infrastructure projects may contain more than 10,000 (potential) requirements.

Each (actual) requirement must be assessed by at least one stakeholder. The requirements manager has to figure out which stakeholders are appropriate for which requirements and needs to assign them accordingly. However, other stakeholders may improve such initial assignments later during the assessment phase. The RM tool notifies all assigned stakeholders via e-mail to assess the requirements they are assigned to.

Table 6.2 shows an example of an initial assignment done by the requirements manager (RM). In this table, each row corresponds to a requirement and each column refers to a stakeholder. Each cell represents a single decision (of a stakeholder) for a stakeholder assignment (to a requirement). At the

Req	RM	PM	RAMS	S(ignal)
R1	{PM}	-	-	-
R2	{PM}	-	-	-
R3	{S}	-	-	-
R4	{S}	-	-	-
R5	{S, PM}	-	-	-
...

Table 6.2.: Initial assignment of stakeholders to requirements done by requirements manager (RM). The dash symbol ("-") indicates that the other stakeholders have not made a decision yet.

Req	RM	PM	RAMS	S(ignal)
R1	{PM}	{PM}	-	-
R2	{PM}	{RAMS}	-	-
R3	{S}	-	-	{S, RAMS}
R4	{S}	-	-	{}
R5	{S, PM}	{S, PM}	-	{S, PM}
...

Table 6.3.: State of assignment during assessment phase

beginning, only the RM proposes assignments of potential stakeholders to requirements based on the manager's expertise and knowledge. For example, the assignment of $\{S, PM\}$ to the requirement $R5$ in the RM column indicates that $R5$ has been initially assigned to the *signal department* (S) and to the *project management department* (PM) by the requirements manager (RM). As only the RM makes assignments in this initialization phase, the values of all other columns remain empty (i.e., are filled with the "-" label) until the assessment phase.

Next, in the assessment phase, the affected stakeholders take a look at each of their assigned requirements in the RM tool and can either accept the requirement and assess it or they can veto the proposed assignment. Additionally, they can also propose an alternative stakeholder for the requirement or suggest (although rarely) an additional stakeholder for the requirement. For the remainder of this chapter, this process is hereinafter referred to as *assignment feedback*. After that, the requirements manager can either accept the veto and assign the requirement to a different stakeholder or decline the veto and reassign the stakeholder to the requirement.

Table 6.3 shows an intermediate state during the assignment phase which demonstrates examples of *assignment feedback* given by the stakeholders *PM* and *S(ignal)*:

- Requirement $R1$ has been accepted by *PM*
- Requirement $R2$ has been vetoed by *PM* and *RAMS* has been proposed by *PM* as alternative stakeholder

Req	RM	PM	RAMS	S(ignal)
R1	{PM}	{PM}	-	-
R2	{RAMS}	{RAMS}	{RAMS}	-
R3	{S, RAMS}	-	{S,RAMS}	{S, RAMS}
R4	{S}	{S}	-	{S}
R5	{S, PM}	{S, PM}	-	{S, PM}
...

Table 6.4.: Final state after assessment phase. Consistent assignment of stakeholders to requirements.

- Requirement *R3* has been accepted by *S(ignal)*, but *RAMS* has been proposed by *S(ignal)* as an additional stakeholder
- Requirement *R4* has been vetoed by *S(ignal)*
- Requirement *R5* has been accepted by all proposed stakeholders

It is important to point out the fact that in the traditional scenario, it is always the main responsibility of the requirements manager to resolve potential conflicts. Typically, this usually involves some personal discussions with the involved stakeholders and some final decisions made by the requirements manager. These final decisions then assure a consistent assignment of all requirements to responsible stakeholders. Table 6.4 presents such a final state where all conflicts have been resolved.

The requirements manager periodically reminds the assigned stakeholders about their unassessed requirements. This process is repeated until all requirements have been assessed and the assessment phase is finished. Thus, the assignment of stakeholders can be considered as a manual configuration process. The outcome of this process is a configuration in terms of a consistent assignment of stakeholders to requirements they are responsible for. In our current implementation, the overall goal is to achieve *consensus* regarding the stakeholder assignment. Future versions of our system include further constraints that have to be taken into account in task allocation tasks as discussed in this chapter.

6.3.2. RM Process with Group Decision Support

The main idea of our novel requirements management approach is to introduce additional stakeholder votes made by *artificial stakeholders* (called *bots*). Additionally, the bots automatically propose stakeholders in the initial phase of the RM process. Furthermore, an intelligent *group decision service* is included in the RM tool to automatically aggregate all votes given by human stakeholders as well as artificial stakeholders. On a technical level, such a *group decision service* represents a *group recommender system* which generates recommendations based on aggregated votes given by group members of a group (i.e., the stakeholders) (Felfernig et al., 2018). Basically, there exist different strategies on how to aggregate votes of group members (Jameson and Smyth, 2007) such as *majority*, *average*, *least misery*, etc. In addition, more sophisticated aggregation functions exist - for further information

regarding preference aggregation functions we refer to Felfernig et al. (2018) and Masthoff (2011). To limit the scope of this chapter, we assume that the group decision service is a simple group recommender using basic aggregation strategies.

The votes of the artificial stakeholders (i.e., bots) are generated by using appropriate *content-based recommendation algorithms* (see Section 6.5). This way, the group decision service helps to replace the traditional (mainly manual) stakeholder assignment process (see Section 6.3.1) with a semi-automatic process. As a key difference to the traditional approach, the group decision service automatically aggregates the decisions of all voters and thereby allows the smart incorporation of additional (automatic) voters, i.e., intelligent recommendation services for stakeholder assignments. From an abstract point of view, the process can be interpreted as a basic configuration process. Like in the traditional RM process (see Section 6.3.1), the outcome of this process represents a consistent assignment of stakeholders to requirements they are responsible for.

Req	GDS	RS1	RM	PM	RAMS	S(ignal)
R1	{PM}	{PM:9}	{PM}	-	-	-
R2	{RAMS}	{RAMS:8, PM:5}	-	-	-	-
R3	{S}	{S:8, RAMS:6}	-	-	-	-
R4	{S}	{S:5}	-	-	-	-
R5	{S}	{S:6}	{S,PM}	-	-	-
...

Table 6.5.: State of assignment with group decision service (GDS) and stakeholder recommendation service (RS1). The recommendation service provides a confidence value which lies in the range between 1 and 10.

Table 6.5 illustrates a possible initial state in the presence of a group decision service (*GDS*) and a stakeholder assignment recommendation service (denoted as *RS1*). In sharp contrast to the assignments made by other stakeholders, the recommendation service does not provide a binary decision for every stakeholder but a confidence value which lies in the range between 1 and 10, whereby a higher number corresponds to more confidence and a lower number corresponds to a lower level of confidence.

The column for the *GDS* shows the result of the group decision service for each requirement, i.e., the aggregated decision of all voters (including humans and bots / algorithms). Note that a clear benefit of the group decision service is that some requirements can already be assessed by the assigned stakeholders, even though they have not yet been proposed / assigned by the requirements manager. In other words, stakeholders are automatically proposed by the bots / algorithms based on their skills in the *initial phase* and can already evaluate their assignment to the requirements. Hence, much assignment effort is taken away in the initial phase from the time-crunched requirements managers

and the initial phase can be speeded up significantly. Moreover, it is necessary to point out that the stakeholders *GDS* (perform aggregation) and *RM* (perform final decision) can be considered to have a special role in this evaluation process, whereas all other stakeholders only occur as voters in the process. Consequently, the major responsibility / task of a *RM* in this process is to review the decision suggested by the *GDS* and to perform the final decision about the assignment of the stakeholders to the requirements.

6.4. Potential Issues of Group Decision Support

The exact behavior of the new system presented in Section 6.3.2 will depend on various factors. Examples of such factors include the *aggregation strategy* used by the group decision service to aggregate the votes (e.g., majority, average, etc.), the individual weight of the voters (e.g., “deciders”/experts count higher than normal stakeholders), and the confidence / trust users have in different recommendation algorithms.

Furthermore, the question arises how conflicting decisions (for example, stakeholder A assigns stakeholder B and B assigns A) can be resolved or supportive advice to manually resolve such conflicts can be given to the voters by the system. Also, inconsistencies and contradictions may occur between the voters in the evaluation of stakeholders. These voters can be other stakeholders and *artificial stakeholders*. In particular, for artificial stakeholders textual explanations can be presented to the group of voters being in conflict. Such textual explanations can then express the concrete reason and arguments for the votes provided by the artificial stakeholders.

Moreover, the prediction quality (i.e., performance) of the *artificial stakeholders* (i.e., the recommenders) plays a major role in the process. In particular, the generated recommendations should be evaluated and examined with respect to completeness. In terms of common information retrieval measures (such as precision and recall), this would, for example, mean that more emphasis should be given to the recall of the results rather than the precision achieved by the recommender. In addition to that, an appropriate recommendation algorithm should also be capable of giving negative indication by telling the *RM* which stakeholders are definitely not suitable to be assigned to a requirement at all. Such a negative indication can be shown as, e.g., RAMS:0. Finally, another important aspect would be to take the availability of stakeholders into account before they get finally assigned to a requirement. This adds another complexity dimension to the underlying basic configuration problem.

6.5. Group Decision Support for Bidding Processes

In this section, a slightly modified version of the aforementioned *RM* process based on *group decision support* (see Section 6.3.2) is described. The description explains the technical implemen-

tation of this process provided by the requirements engineering platform OPENREQ!LIVE³ which has been developed within the scope of the OPENREQ EU Horizon 2020 research project. At the current stage, the implementation is already in use, however, still ongoing and ready to be further enriched with additional features. The remainder of this section describes the current status of the existing implementation.

In the initial phase, the requirements manager (RM) is asked by the system to propose suitable stakeholders for each requirement. As already described in Section 6.3.2, a content-based recommender system (RS1) helps the *RM* to find stakeholders based on keywords extracted from former requirements these stakeholders have solved. Thereby, on an abstract level, the automated stakeholder-recommendation algorithm (of RS1) can be interpreted as a text classification task (Ikonomakis et al., 2005) where the recommendation algorithm exploits several *natural language processing* (Ryan, 1993; Winkler and Vogelsang, 2016) techniques in order to correctly classify stakeholders suitable for a given requirement.

The algorithm automatically extracts relevant keywords from the title and description text of all former requirements to which a stakeholder was assigned, in order to build a user profile for the respective stakeholder. First, the title and description text is cleaned by removing special characters (such as “:”, “;”, “,”, “#”, etc.). Next, the text is split into tokens (which, basically, represent the words in the text) and *stop words*⁴ are removed. After applying *part-of-speech tagging*, tokens / words of classes (such as verbs, adjectives, or numbers) that are most probably irrelevant to be used as keywords, are removed. Finally, the remaining tokens of each former requirement (which was assigned to the stakeholder) are merged together into a single user profile.

By applying the same procedure to new requirements, keywords for new requirements are extracted as well. Given the keywords of a new requirement and the user profiles of the individual stakeholders, a similarity between a new requirement and a stakeholder is calculated for every stakeholder provided that the stakeholder has been assigned to an (already completed) requirement in the past. Formula 6.1 shows the *Dice coefficient formula* (Jannach et al., 2010) which is a variation of the Jaccard coefficient and used to compute the similarity between a stakeholder and a requirement. The similarity is measured by comparing the overlap of the keywords of the stakeholder’s user profile (denoted as U_a) and the relevant keywords of the respective requirement (denoted as r_x) with the total number of keywords appearing in U_a as well as r_x .

$$sim(U_a, r_x) = \frac{2 * |keywords(U_a) \cap keywords(r_x)|}{|keywords(U_a)| + |keywords(r_x)|} \quad (6.1)$$










Stakeholders who are most similar to a given requirement are suggested by the content-based rec-

³OPENREQ!LIVE: <https://github.com/OpenReqEU/openreq-live>

⁴Examples of *stop words* include prepositions (e.g., “in”, “on”, “at”, etc.) and articles (e.g., “the”, “a”, “an”).

ommender to the *RM*. This way, the initial phase can be speeded up and the chance of overseeing suitable stakeholders for requirements at this early stage of the process, is decreased. In the next step, the OPENREQ!LIVE system shows a list of the initially assigned stakeholders for each requirement. Stakeholders who are assigned to a requirement can either accept or reject their assignment. In addition, the assignments of the stakeholders for the requirement can be evaluated by all stakeholders.

Stakeholders assigned to Requirement #3:

	✖ Appropriateness		🕒 Availability		Result	
	Your	Average	Your	Average		
 Martin St.  	10	10.0	8	8.0	9.0	✖
 Muesluem At. Accepted  	9	8.0	7	8.5	8.3	✖
 Ralph Samer  	8	8.5	4	6.0	7.3	✖

Assign stakeholders:

Enter name... + ASSIGN

Figure 6.1.: Evaluation of stakeholders in OPENREQ!LIVE. Each stakeholder-assignment is evaluated by two evaluation dimensions (appropriateness and availability). The utility value of an evaluated stakeholder is calculated by using Formula 6.2.

This evaluation of a stakeholder-assignment is done based on the criteria *appropriateness* and *availability* (see Figure 6.1). Both criteria are interpreted as *evaluation dimensions* and stakeholders are evaluated based on both dimensions. Furthermore, an assigned stakeholder can also propose the assignment of further stakeholders to the requirement. These newly assigned stakeholders can then be evaluated again. After a new vote has been given, the *group decision service* (GDS) is triggered to compute a utility value for the rated stakeholder. Formula 6.2 shows the calculation of the utility value of an evaluated stakeholder s , whereas D represents the set containing both dimensions, i.e., $D = \{Appropriateness, Availability\}$.

$$utility(s, r) = \frac{\sum_{t \in T} \frac{\sum_{d \in D} eval(s, r, d, t) \cdot weight(d)}{\sum_{d \in D} weight(d)}}{|T|} \quad (6.2)$$

The formula describes the stakeholder s to be voted by other stakeholders and T represents the set of stakeholders $t \in T$ who evaluated s . More formally expressed, T is a set which contains the stakeholders (including s) who evaluated stakeholder s (i.e., $T \subseteq S$). Furthermore, the OPENREQ!LIVE platform allows the RM to define different importance levels for both dimensions. In Formula 6.2, the importance of a dimension $d \in D$ is expressed by the function $weight(d)$. Moreover, $eval(s, r, d, t)$ refers to the dimension-specific rating given by stakeholder t for stakeholder s for the requirement r . Finally, the result of $utility(s, r)$ represents the aggregated utility of a stakeholder s for requirement r .

Once all assignments have been evaluated by a sufficient number of stakeholders, a stable state of the assignment utilities is achieved. The utility values are then used as main feedback source for the RM to make the final decision about which stakeholder(s) should be assigned to the requirement.

6.6. Conclusion and Future Work

Conclusion. In this chapter, we discussed common application scenarios of requirements engineering in the context of industry projects. These scenarios range from traditional requirements management processes where the assignment process of stakeholders is solely controlled by the requirements manager, to more sophisticated automated approaches where the involvement of the requirements manager is reduced to a minimum. The latter represents a basic configuration service which includes *artificial stakeholders* as additional voters and a *group decision support system* as a vote aggregation component in the evaluation of stakeholder assignments to requirements. On the basis of this scenario we showed how these two components can be applied in order to improve the requirements management process such that the overall effort and the chance of overseeing stakeholders suitable for requirements can be reduced for the time-crunched requirements managers.

Future Work. As bidding processes can be seen as repetitive processes, mechanisms capable of learning stakeholder weights and taking individual expertise levels of stakeholders into account can be considered as potential ideas regarding future work. Moreover, the set of existing evaluation dimensions can be further extended such that more fine-grained control is given to the evaluation process as well as to the *group decision service*. Additionally, the concept of *liquid democracy* can be integrated into the evaluation process (Kahng et al., 2018). This way, stakeholders who do not have sufficient knowledge about the details of a requirement can easily delegate their votes to more well-informed and experienced experts. With respect to conflicting decisions (see Section 6.4), future work should also include mechanisms to automatically resolve such conflicts or mechanisms, providing supportive advice to the voters, showing them how they can manually resolve such conflicts. Furthermore, the configuration approach can be extended with further constraints taking resource management aspects of stakehold-

ers into account, in order to optimize the overall allocation of human resources in release planning. Finally, there is also still plenty of room for improvement regarding the extraction of keywords used by the discussed content-based recommender (i.e., *artificial stakeholder*). For example, a more descriptive and characteristic representation of the keywords can be obtained by using more advanced content-based approaches such as *Latent Semantic Analysis* (Landauer et al., 1998; Mikolov et al., 2013) or approaches supporting *word embeddings* (Lau and Baldwin, 2016; Mikolov et al., 2013).

Towards Utility-based Prioritization of Requirements in Open-Source Environments

This chapter is based on the ideas and concepts documented in Felfernig et al. (2018). Major parts in terms of literature research and the design of algorithmic approaches have been provided by the author of this thesis.

7.1. Abstract

Requirements engineering in open-source projects such as ECLIPSE faces the challenge of having to prioritize requirements for individual contributors in a more or less unobtrusive fashion. In contrast to conventional industrial software development projects, contributors in open-source platforms can decide on their own which requirements to implement next. In this context, the main role of prioritization is to support contributors in figuring out the most relevant and interesting requirements to be implemented next and thus avoid time-consuming and inefficient search processes. In this chapter, we show how utility-based prioritization approaches can be used to support contributors in conventional as well as in open-source requirements engineering scenarios. As an example of an open-source environment, we use BUGZILLA. In this context, we also show how dependencies can be taken into account in utility-based prioritization processes.

7.2. Introduction

In software projects, resources are typically limited which requires the prioritization of requirements (Lehtola et al., 2004). Prioritization is often interpreted as a part of *strategic planning* where the focus is to select and prioritize requirements that should be included in releases (long-term release planning)

(Ameller et al., 2017; Ruhe and Saliu, 2005). Decision support in prioritization is extremely important since especially when dealing with large assortments of requirements, manual prioritization processes tend to become very costly (Alenezi and Banitaan, 2013; Xuan et al., 2012). In this context, suboptimal prioritizations trigger time wasting due to the implementation of unimportant requirements.

There are *two basic approaches* to prioritize requirements – for an in-depth related analysis we refer to Achimugu et al. (2014). *First*, requirements prioritization can be interpreted as an *optimization task* where the overall objective is to identify the *middle ground*, i.e., an aggregation of individual prioritizations into a global prioritization that reflects the least possible level of dissimilarity from all stakeholder-individual prioritizations (Kifetew et al., 2017). *Second*, in contrast to approximating individual prioritizations on the basis of optimization functions, *utility-based approaches* focus on (1) establishing agreement with regard to the evaluation of individual requirements and (2) thereafter determining prioritizations (Adomavicius et al., 2011; Huang, 2011; Wiegers, 2003).

Prioritizations following the *optimization approach* are determined on the basis of individual prioritizations of stakeholders. When following a *utility-based approach*, preferences of stakeholders are first aggregated and a prioritization is determined thereafter. In the line of basic approaches to determine group recommendations (Felfernig et al., 2018), the first approach is based on *aggregated prioritizations* where stakeholder-individual prioritizations are known and a recommendation minimizes dissimilarities between the given prioritizations (preferences). The second approach is based on *aggregated models* where stakeholder requirement evaluations are aggregated first and a prioritization is determined on the basis of a group profile (model) derived from requirements evaluations.

Aggregated models have the advantage that stakeholders are encouraged to focus their evaluations on specific relevant aspects of a requirement (e.g., dimensions such as *profit*, *risk*, and *effort*) and thus contribute to stable preferences and a higher degree of consensus (Stettinger et al., 2015). Aggregated prioritizations trigger scalability issues since each stakeholder has to provide, for example, a ranked list of requirements as input for the optimization process. Furthermore, due to the computational complexity of the underlying problem, an optimal solution cannot be guaranteed and is often only approximated on the basis of local search algorithms (Kifetew et al., 2017; Tonella et al., 2013; Zhang et al., 2007). Utility-based approaches as discussed in this chapter focus on evaluations of individual requirements on the basis of different evaluation dimensions (e.g., *profit*, *effort*, and *risk*). This way, stakeholders can focus on evaluating requirements they have knowledge about and the focus of prioritization is first on establishing *consensus* and thereafter on figuring out the most relevant prioritizations (Stettinger et al., 2015).

Different algorithmic approaches can be used to support requirements prioritization – for an overview, see, for example, Achimugu et al. (2014). Examples thereof are constraint-based reasoning (Tsang,

1993), incremental preference learning (Perini et al., 2013), evolutionary algorithms (Kifetew et al., 2017), machine learning (Alenezi and Banitaan, 2013; Tian et al., 2015), and pairwise preference-based decision making (Saaty and Vargas, 2000). *Optimization-based approaches* focus on minimizing the distance between the preferences of individual stakeholders (e.g., in terms of the distance between individual prioritizations and the prioritization determined by the optimization approach). A similar problem also solved on the basis of optimization approaches is the *next release problem* (Bagnall et al., 2001; Xuan et al., 2012) where a subset of a given set of requirements has to be selected in such a way that predefined cost limits are taken into account and the chosen set of requirements represents the optimum choice in terms of criteria such as market value. The focus in this context is more to identify subsets of requirements but not to prioritize a given list of requirements. In contrast to next release problems, prioritization tasks in open-source scenarios do not necessarily require (and often do not allow) a global optimum but more focus on relevant recommendations for individual stakeholders. Also in contrast to existing release planning tasks, developers in open-source scenarios in most of the cases do not explicitly define their preferences, i.e., preferences have to be derived from given interaction data (in our case, interaction data collected by BUGZILLA¹).

Utility-based prioritization based on *multi-attribute utility theory* (Dyer, 1997) can be implemented in different variants. First, requirements are simply evaluated with regard to a set of predefined interest dimensions and the overall utility of a requirement is determined as a sum of interest dimension specific utilities. Second, weights can be introduced to emphasize on specific interest dimensions (e.g., a lower risk is more important than high profits). Third, stakeholders can be enabled to define their personal evaluations and utility-based approaches should then be able to aggregate these evaluations and take into account stakeholder weights. Stakeholder weights can be interpreted as "global", i.e., there is a global weighting of stakeholders independent of a specific dimension or requirement. If weights are interpreted as "local", the importance of a stakeholder can be defined on the level of individual requirements or dimensions. Utility-based prioritization can also be implemented on the basis of an *analytic hierarchy process* (AHP) (Karlsson and Ryan, 1997). A major disadvantage of this approach is that requirements have to be evaluated pairwise which does not scale well when the number of requirements increases.

Prioritization criteria differ depending on the requirements engineering scenario. The criteria *effort*, *risk*, and *profit* are often used in settings where a group of stakeholders engaged in the same project is in charge of completing a prioritization task (Achimugu et al., 2014). In contrast, in open-source settings, developers are in most of the cases engaged in different projects and also work for different companies. In such scenarios, prioritization is less focusing on establishing consensus between individual stakeholders but more on supporting stakeholders in identifying requirements of relevance to them and to prioritize the important ones by also taking into account global criteria. Examples of

¹BUGZILLA: <https://www.bugzilla.org>

criteria in such scenarios are *personal expertise* of a developer and *importance* of a requirement for the community of the stakeholder and the open-source community as a whole. Thus, open-source platform related prioritization processes completely differ from conventional software projects. A major focus of this chapter is to introduce prioritization concepts especially applicable in open-source development contexts.

BUGZILLA is an open-source based issue tracking system which supports users from different geographical locations to report their findings with regard to a given set of software components. Users can submit textual descriptions of issues and corresponding meta-information, for example, associated components, keywords, and dependencies. Reported issues can be selected by contributors to work on. In BUGZILLA, issues can be requirements but also reported bugs. Distinguishing between these can be performed on the basis of a meta-attribute (issue type) that can be specified for BUGZILLA issues. There are different related approaches to support machine learning based requirements prioritization. The approaches operate on datasets including historical data of previous requirement (bug report in BUGZILLA) selections and try to predict future requirement selections thereof. Utility-based prioritization can be used in interactive scenarios (stakeholders are engaged in an interactive prioritization process) as well as in scenarios where requirements are recommended but no further stakeholder interaction is needed for determining a prioritization.

The *contributions* of this chapter are the following. We provide an overview of different application scenarios of utility-based requirements prioritization and discuss specific aspects of requirements prioritization in open-source projects. For scenarios that include dependencies between requirements, we show how such dependencies can be taken into account on the basis of the concepts of model-based diagnosis (Felfernig et al., 2012; Reiter, 1987). With this approach, we tackle the following *research gaps*. In contrast to existing prioritization and release planning approaches, we introduce model-based diagnosis concepts that also support re-prioritization and re-planning while not completely omitting already existing stakeholder preferences which is still an open issue in most of the existing prioritization and release planning approaches (these approaches focus on taking into account dependencies but do not support the aforementioned re-prioritization and re-planning scenarios). Furthermore, we present a first version of a user interface developed to support prioritization tasks in open-source environments – in our case, for BUGZILLA users. This approach has the potential to reduce the workload of developers in open-source platforms by automatically proposing requirements that should be implemented next instead of forcing users to analyze in detail a large number of requirements. Furthermore, the introduced utility-based approach does not require the "manual" evaluation of individual requirements but automatically derives utility models by analyzing given interaction logs taking into account interaction data such as *number of comments* related to a requirement. This is a major difference compared to existing requirements prioritization approaches which do not support automated prioritization based on background data. Finally, we discuss issues for future work to further advance the

state of the art in utility-based prioritization.

The remainder of this chapter is organized as follows. In Section 7.3, we introduce variants of implementing *utility-based prioritization*. Thereafter, we introduce our variant of utility-based prioritization implemented for the BUGZILLA environment and provide a sketch of a related BUGZILLA user interface (Section 7.4). In Section 7.5, we show how to extend utility-based prioritization in such a way that dependencies between requirements can be taken into account. The chapter is concluded with a discussion of future work in Section 7.6.

7.3. Utility-based Prioritization

Utility-based prioritization allows stakeholders to prioritize a requirement with regard to different interest dimensions $D = \{d_1, d_2, \dots, d_n\}$. Examples of such interest dimensions are *profit*, *risk*, and *effort*. Utility-based prioritization is based on the idea to first evaluate each requirement with regard to the set of interest dimensions (see Table 7.1) and thereafter calculate the individual utility of each requirement (see Formula 7.1). In general, the *priority* is associated with the *utility* of a requirement r which results from its total contributions to all of each individual interest dimensions d (denoted as $contribution(r, d)$) combined with the corresponding importance weights of individual interest dimensions (denoted as $weight(d)$).

interest dimension	r_1	r_2	r_3
profit	10	5	4
risk	7	2	8
effort	2	3	7

Table 7.1.: Contribution of requirements $R = \{r_1, r_2, r_3\}$ to the interest dimensions $D = \{profit, risk, effort\}$.

interest dimension	weights
profit	0.3
risk	0.5
effort	0.2

Table 7.2.: Predefined weights for the interest dimensions $D = \{profit, risk, effort\}$.

$$utility(r, D) = \sum_{d \in D} (contribution(r, d) \times weight(d)) \quad (7.1)$$

Applying Formula 7.1 to the entries in Tables 7.1 and 7.2 results in the ranking depicted in Table 7.3 (the higher the utility with regard to the given interest dimensions, the higher the corresponding priority of the requirement).

requirement	r_1	r_2	r_3
utility	6.9	3.1	6.6
priority (ranking)	1	3	2

Table 7.3.: Ranking of requirements with static weights.

In the previous example, the evaluation of requirements with regard to interest dimensions and the weighting of interest dimensions are assumed to be predefined (e.g., by a single stakeholder). However, requirements prioritization is often a group decision process (Felfernig et al., 2018) where different stakeholders are evaluating requirements (see, e.g., Table 7.4) and define importance weights with regard to interest dimensions (see, e.g., Table 7.5²). Both, stakeholder-individual evaluations of interest dimensions and importance weights have to be aggregated. Formula 7.2 shows the aggregation of stakeholder-individual evaluations of requirements where S refers to the set which includes all m stakeholders (i.e., $S = \{s_1, s_2, \dots, s_m\}$).

interest dimension	r_1				r_2				r_3			
	s_1	s_2	s_3	AVG	s_1	s_2	s_3	AVG	s_1	s_2	s_3	AVG
profit	5	2	2	3.0	5	1	2	2.7	2	2	6	3.3
risk	3	3	4	3.3	2	5	6	4.3	3	2	2	2.3
effort	2	3	2	2.3	3	4	2	3.0	5	6	2	4.3

Table 7.4.: Contribution of requirements $R = \{r_1, r_2, r_3\}$ to dimensions $D = \{profit, risk, effort\}$ (defined by stakeholders $S = \{s_1, s_2, s_3\}$).

$$contribution(r, d, S) = \frac{\sum_{s \in S} eval(d, r, s)}{|S|} \quad (7.2)$$

Formula 7.3 shows how to aggregate the stakeholder-specific importance weights (denoted as $w(d, s)$) which are related to individual interest dimensions d . Previous calculations did not take into account potential different degrees of stakeholder expertise, for example, a stakeholder s_a could have more expertise with regard to estimating the market potential of a requirement in terms of profit as estimating the corresponding development efforts. To take into account this aspect, Formula 7.3 includes a factor that represents the *expertise* of a stakeholder s with regard to a specific interest dimension d .

²In this example, expertise(d,s) was assumed to be 1 for all stakeholders and dimensions.

$$weight(d, S) = \frac{\sum_{s \in S} w(d, s) \times expertise(d, s)}{|S|} \quad (7.3)$$

Similar to the basic approach, the utility of a requirement (Formula 7.4) is determined as a combination of the contributions of a requirement to the given interest dimensions and related interest dimension importance evaluations of stakeholders.

$$utility(r, D, S) = \sum_{d \in D} (contribution(r, d, S) \times weight(d, S)) \quad (7.4)$$

stakeholder	s_1	s_2	s_3	weights
profit	0.5	0.3	0.6	0.47
risk	0.3	0.6	0.3	0.4
effort	0.2	0.1	0.1	0.13

Table 7.5.: Preferences of stakeholders $S = \{s_1, s_2, s_3\}$ with regard to the interest dimensions $D = \{profit, risk, effort\}$.

The result of applying Formulae 7.2–7.4 to the evaluation data contained in Tables 7.4 and 7.5 is depicted in Table 7.6.

requirement	r_1	r_2	r_3
utility	3.03	3.38	3.03
priority (ranking)	2	1	2

Table 7.6.: Ranking of requirements with group weights.

7.4. Utility-based Prioritization in BUGZILLA

In Section 7.3, we took a look at different variants of utility-based prioritization. These variants were discussed on the basis of interest dimensions (evaluation criteria) typically occurring in software projects where a group of stakeholders is in charge of jointly defining and prioritizing requirements. In this section, we focus on open-source scenarios where individual users (e.g., contributors in an open-source platform) follow their individual interests regarding requirements without necessarily taking into account the preferences of other users. This can be considered a major difference compared to conventional software projects where stakeholders commonly develop a "global" prioritization (see Section 7.3). We now show how utility-based prioritization can be applied in such contexts.

Table 7.7 represents a BUGZILLA-specific evaluation of requirements (bugs) with regard to the set of interest dimensions $\{cc, geritt, blocker, comments\}$. In this context, cc is the number of contributors who are in the :cc list of bug-related emails, $geritt$ is the number of bug-related GERITT³ changes, $blocker$ is the number of dependent bugs (dependent requirements), and $comments$ refers to the number of bug-related comments. These interest dimensions do need to be evaluated manually as it is often the case in other scenarios (Laurent et al., 2007; Shao et al., 2017) but can directly be determined from corresponding user interaction data.

Formula 7.5 supports the calculation of the contribution of a requirement r to a specific interest dimension d . In sharp contrast to the previous scenarios, the contribution is not directly specified by stakeholders but derived from BUGZILLA specific information (e.g., #comments related to a requirement).

$$contribution(r, d) = eval(r, d) \tag{7.5}$$

Formula 7.6 supports the determination of the expertise of a stakeholder which is represented in terms of the similarity between the keywords stored in the stakeholder profile and those extracted from the requirement description and corresponding meta-information (e.g., name of associated component / system). The similarity between requirement-related keywords, meta-information, and contributor profile information is interpreted as *expertise* (see Formula 7.6).

$$weight(r, s) = expertise(r, s) \tag{7.6}$$

In the line of the previously discussed utility functions, the overall utility of a requirement is interpreted as a combination of (1) the contributions of a requirement to a set of interest dimensions and (2) the expertise level of a stakeholder (in this context interpreted "globally", i.e., not on the level of individual interest dimensions).

$$utility(r, s) = \sum_{d \in D} contribution(r, d) \times weight(r, s) \tag{7.7}$$

interest dimension	r_1	r_2	r_3
cc	5	2	2
geritt	3	3	4
blocker	2	3	2
comments	2	3	2

Table 7.7.: Contribution of requirements (bugs) $R = \{r_1, r_2, r_3\}$ to the interest dimensions $D = \{cc, geritt, blocker, comments\}$.

³A code reviewing tool – <https://www.gerritcodereview.com>.

stakeholder	s_1
r_1	0.5
r_2	0.3
r_3	0.2

Table 7.8.: Expertise of stakeholder s_1 with regard to the requirements $\{r_1, r_2, r_3\}$ determined, for example, on the basis of the similarity between the stakeholder profile and information associated with a requirement.

requirement	r_1	r_2	r_3
utility	6.0	3.3	2.0
priority	1	2	3

Table 7.9.: Ranking of BUGZILLA bugs with static weights.

7.5. Taking into Account Dependencies

Blocking Factor. Utility-based recommendation approaches per se do not explicitly take into account dependencies between requirements (e.g., requirement A must be implemented before requirement B). In the discussed open-source prioritization scenario, this aspect is taken into account by prioritizing requirements on the basis of the number of related dependencies, i.e., the higher the number of requirements dependent from a requirement x , the higher the "global" relevance of x . In this context, the *blocking factor* (i.e., how many requirements depend on the implementation of requirement x) can be considered as interest dimension that has an impact on prioritization. In other words, this requirement should be implemented as soon as possible since it otherwise blocks the implementation of other requirements. This approach can also be applied in software development scenarios where a group of stakeholders (e.g., an in-house software development project) is in charge of prioritizing requirements. Such an approach helps to avoid situations where prioritizations violate dependency constraints.

Automated Repair. An alternative approach is to apply repair mechanisms from model-based diagnosis (Felfernig et al., 2012) that help to adapt already determined prioritizations in such a way that all defined dependencies are taken into account. In the following, we will shortly sketch our approach. In order to trigger a diagnosis process, we are in the need of a pre-defined set of dependencies between requirements (denoted as $DEP = \{dep_1, dep_2, \dots, dep_n\}$). Furthermore, we assume that a prioritization (represented as sequence) $P = [p_1, p_2, \dots, p_m]$ determined by a utility-based prioritization approach is *inconsistent* with the given set of dependencies. In order to apply model-based diagnosis, we assume that both, the pre-defined set of dependencies and the requirement prioritization is represented in terms of constraints (Tsang, 1993), for example, $DEP = \{dep_1 : r_3 < r_1, dep_2 : r_3 < r_2\}$ and

$P = \{p_1 : r_1 < r_2, p_2 : r_2 < r_3, p_3 : r_3 < r_4, p_4 : r_4 < r_5, p_5 : r_5 < r_6\}$. As can be easily seen, $DEP \cup P$ is inconsistent. As variable domains we assume [1..#requirements].

Following the principles of model-based diagnosis (Felfernig et al., 2012; Reiter, 1987), we need to detect all *minimal conflicts* (Junker, 2004) induced in P by the dependencies defined in DEP . In this context, a conflict set is defined as follows.

Definition: Conflict Set (CS). A conflict set $CS \subseteq P$ is a set of individual prioritization elements that are inconsistent with the elements of DEP , i.e., $inconsistent(CS \cup DEP)$. A conflict set CS is minimal if $\neg \exists CS' : CS' \subset CS$.

On the basis of a set of identified minimal conflict sets, a corresponding diagnosis includes a set of prioritization elements in P that have to be adapted such that the consistency of $DEP \cup P$ is restored (see the following definition).

Definition: Diagnosis (Δ). A diagnosis $\Delta \subseteq P$ is a set of individual prioritization elements that have to be deleted from P such that $consistent(P - \Delta \cup DEP)$.

In our example, $CS : \{p_2\}$ is the only conflict induced in P by the dependencies defined in DEP . CS is minimal, i.e., we need to adapt only one of the prioritization elements in CS such that a global prioritization can be found that is consistent with the elements in DEP (Junker, 2004). A corresponding diagnosis Δ is $\{p_2\}$. In our example, we could decide to replace $p_1 : r_1 < r_2$ with the corresponding repair $r_1 > r_2$. This is a repair action that helps to restore the consistency of $DEP \cup P$.

Our approach to the repair of inconsistent prioritizations can be used for both, *interactive prioritization* where stakeholders receive feedback on the consistency of prioritizations, and *automated prioritization* where repairs for inconsistent prioritizations are determined in an automated fashion. Important issues to improve our approach are discussed in the following.

7.6. Conclusion and Future Work

Conclusion. In this chapter, we showed how to support utility-based requirements prioritization. These scenarios range from *single user prioritization* where one stakeholder is in charge of completing prioritization tasks to *group-based prioritization* where the preferences / evaluations of different group members have to be taken into account. On the basis of these scenarios, we showed how utility-based prioritization can be applied in the context of open-source development projects. In this context, we sketched our initial implementation currently provided in the BUGZILLA environment. This implementation serves as a first version to support requirements prioritization in BUGZILLA.

Future Work. Since prioritization is a repetitive process, we will include mechanisms that are capable of learning stakeholder weights and also the weights of individual requirements. This approach will help to further increase the prediction quality of prioritizations in terms of the probability that stakeholders accept the proposed prioritizations. Moreover, we will analyze which further features (interest dimensions) are useful to improve prediction quality. For example, the number of redundant bugs (issues) can be a further important relevance indicator.

A major challenge in requirements prioritization is the provision of persuasive user interfaces that increase the preparedness of stakeholders to actively engage in requirements engineering processes (Atas et al., 2017). Consequently we will focus on a further extension / improvement of the existing BUGZILLA requirements prioritization user interface. Furthermore, we will analyze in which way recommended prioritizations have to be *explained* to support specific group decision goals, such as *consensus*, *fairness*, and *decision quality* (Du and Ruhe, 2009; Felfernig et al., 2018). Finally, we will conduct a detailed empirical study regarding the impact of our prioritization approaches in the ECLIPSE community.

Towards Issue Recommendation for Open-Source Communities

This chapter is based on the results documented in Samer et al. (2019). Imperative parts of this chapter, such as literature research, algorithmic approaches and the scientific evaluation of test results have been provided by the author of this thesis. For this work, we received the runner-up for the Best Student Paper Award at the International Conference on Web Intelligence in 2019 (WI 2019).

8.1. Abstract

In open-source software development, a major challenge is the prioritization of new requirements as well as the identification of responsible developers for their implementation. Unlike conventional industrial software development, where requirements engineers have to explicitly define *who implements what*, in the context of open-source development, developers (contributors) usually decide on their own which requirements to implement next. Contributors have to deal with a huge number of requirements where the recognition of the most relevant ones often becomes a crucial task with a high impact on the success of a software project. This fact defines our major motivation for the development of a prioritization tool for the ECLIPSE community which recommends relevant requirements (issues / bugs) to open-source developers. Our tool uses real-world data from ECLIPSE in order to build a prediction model. We trained and tested our tool with different classifiers such as *Naïve Bayes* (representing our baseline), *Decision Tree*, and *Random Forest*. The evaluation results indicate that the Random Forest classifier correctly predicts issues with a precision of 0.88 (f1-score: 0.68).

8.2. Introduction

In software development projects, resources are in most of the cases limited which requires the prioritization of requirements (Lehtola et al., 2004). Prioritization can be regarded as a part of strategic

planning with the focus of selecting and prioritizing requirements that should be included in upcoming releases (Ameller et al., 2017; Ruhe and Saliu, 2005; Ninaus et al., 2014; Anvik et al., 2006; Alenezi and Banitaan, 2013). Efficient support of prioritization decisions is extremely important since especially when dealing with large assortments of requirements, manual prioritization processes tend to become very costly (Xuan et al., 2012). In this context, suboptimal prioritizations are responsible for the implementation of unimportant requirements.

In open-source software development, developers (contributors) are spatially distributed and nevertheless need access to the current requirements (issues / bugs)¹ of relevance. As a consequence, in many cases an issue repository is used where contributors select issues to work on. This process has to be supported with a centralized repository that is accessible to every contributor. Due to huge amounts of issues as well as contributors, the management of a centralized issue repository quickly gets a herculean task (Anvik et al., 2006). Deciding about the relevance of an issue as well as identifying the most appropriate contributors are major challenges.

The aim of our work is to better support developers (contributors) during the interaction with issue tracking environments (in our context, BUGZILLA²) and to bring issue tracking technologies to the next level. To achieve this goal, we apply recommendation technologies (Jannach et al., 2010) to generate personalized lists of issues to be shown to contributors of an issue tracking environment. Figure 8.1 includes a screenshot of our ECLIPSE³ plugin for issues in BUGZILLA that provides a recommendation as prioritized list of potentially relevant issues. This functionality helps to reduce time efforts needed to identify the next relevant issues to work on, since contributors do not need to search manually for the next relevant issue.

The improvement of issue management processes in open-source software development scenarios is also in the focus of other research contributions. Alenezi and Banitaan (2013) introduce machine learning techniques such as *Naïve Bayes*, *Decision Trees*, and *Random Forest* to calculate the predictions of issue priorities. An approach that uses classification-based machine learning for the calculation of the priority level of individual requirements is introduced by Tian et al. (2015) - the presented approach exploits besides the requirement data itself also requirement-related information. In the line of the work of Tian et al. (2015), Menzies and Marcus (2008) present an approach with the aim to predict the degree of severity of reported software defects in NASA software projects. The presented machine learning approach estimates the severity level of a defect. Felfernig et al. (2018) developed a basic utility-based prioritization approach for issues in BUGZILLA which takes into account relevant meta-information of the issues such as the number of issue-related comments or the number of dependent bugs. Furthermore, Alenezi et al. (2013) introduce a content-based approach that is used for the

¹In the following, we assume that the term *issue* entails the concept of *bugs*.

²BUGZILLA: <https://www.bugzilla.org>

³ECLIPSE IDE: <https://www.eclipse.org/ide>

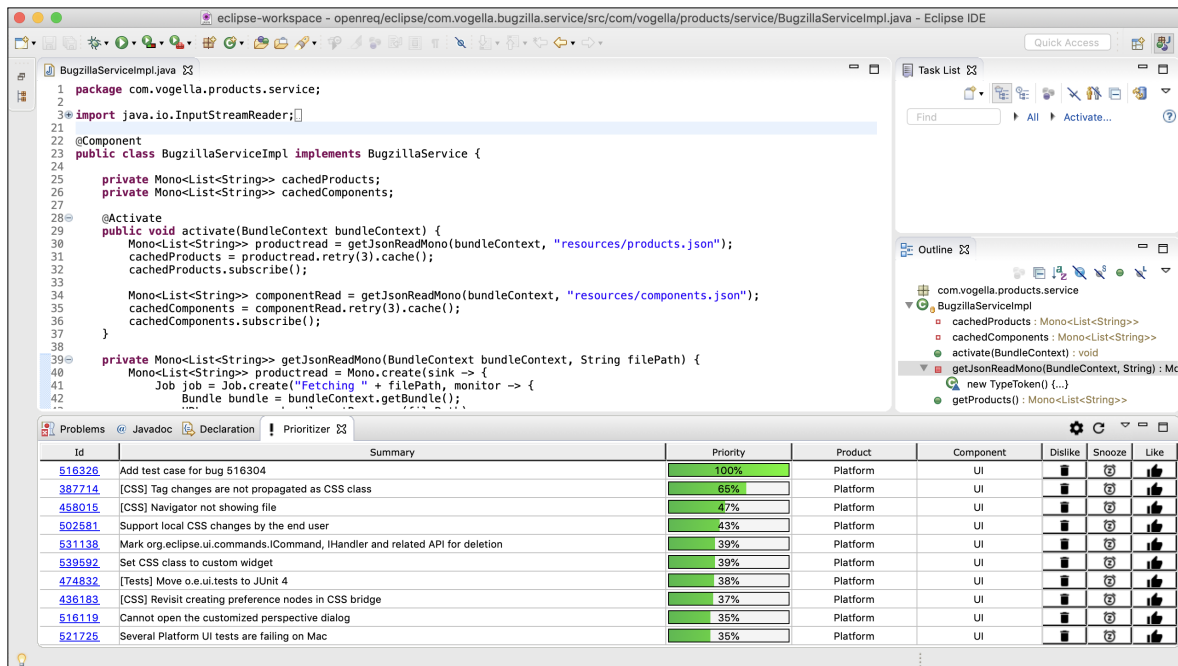


Figure 8.1.: ECLIPSE plugin for the contributor-specific prioritization of bugs.

automated prediction of relevant bugs based on the experience of the contributor. A slightly different approach is presented by Stanik et al. (2018). The presented classification-based approach identifies bugs that are (a) feasible for new contributors and (b) increase the retention rate of the contributors working on it. To achieve this, the approach is tailored towards the understanding on which issues new contributors (with an often low level of related experience) should work on next.

In comparison to related work, the major focus of the work presented in this chapter is to identify personalized recommendations of issues that are tailored towards the current topics of interest of individual contributors. Our approach offers recommendations for the mainstream and does not focus on a specific subset of contributors as presented in the work of Stanik et al. (2018). The recommended list of issues represents the most relevant bugs in descending order and lets the contributors decide about the bugs to work on.

The major contributions of our work are the following. First, we introduce a content-based recommendation approach based on supervised classification that generates issue prioritizations. On the basis of such prioritizations, contributors can decide on their own on which issues they want to work on. Furthermore, we have developed a plugin for ECLIPSE in terms of a graphical user interface for the ECLIPSE developer community (see Figure 8.1). Finally, in order to evaluate our approach, we used the classifiers *Multinomial Naïve Bayes* (our baseline), *Decision Tree*, and *Random Forest* and evaluated these with issue datasets from three different open-source projects (ECLIPSE, MOZILLA,

LIBREOFFICE). The evaluation results for all three datasets indicate that the classifier based on the Random Forest algorithm is able to achieve a good prediction quality in terms of precision compared to our baseline.

The remainder of this chapter is structured as follows. In Section 8.3, we introduce our recommendation approach that supports the prioritization of issues for individual stakeholders. In Section 8.4, we analyze the predictive quality of the proposed approach. A discussion of ideas for future work is given in Section 8.5. The chapter is concluded with Section 8.6.

8.3. Methodology

Our work is dedicated to the development of a recommender system (Jannach et al., 2010) for developers working on open-source projects. The major objective of the recommender system is to recommend issues to these developers that are relevant to them.

8.3.1. Datasets

We used three datasets from different open-source projects (ECLIPSE, MOZILLA, LIBREOFFICE) to train and evaluate our system. The mentioned open-source projects share in common that they use the same issue tracking software BUGZILLA which provides a *web API* to download the project's issue data. The BUGZILLA issue tracking system is used by these communities to coordinate and maintain their tasks. The dataset of each project contains bugs / issues from the last 10 years. We only collected *resolved* issues which are issues that have been assigned to a (single) contributor. Every *resolved* issue can be either marked as **FIXED** (*this issue is fixed*), **INVALID** (*the reported problem is not a bug / issue*), **WONTFIX** (*the bug cannot be fixed*), **DUPLICATE** (*the issue is a duplicate of another existing issue*), or **WORKSFORME** (*the bug / issue seems not to be an issue as it cannot be reproduced*). Our work focused on *resolved* issues which are marked as **FIXED** because only such issues can be regarded as to be successfully completed by their assigned developers. We also excluded non-resolved issues (e.g., open issues, work in progress, etc.) from the dataset as these issues are not suitable for the purpose of learning a recommendation model. Our ECLIPSE dataset contains 141,117 issues (resolved by 2,758 contributors), the MOZILLA dataset consists of 751,961 issues (resolved by 6,962 contributors), and the LIBREOFFICE dataset contains 47,542 issues (resolved by 638 contributors) (see Table 8.1). Each issue consists of a *title*, a *textual description*, an *assignee* (who represents the contributor to whom the issue was assigned) and much further meta information such as the issue's *status*, *attached files* (e.g., screenshots), or *links* to other related issues.

In BUGZILLA, issue reporters (i.e., the users who report bugs / issues) must assign their newly created issue to a (single) component which further belongs to a product. The purpose of a *component* is to group issues which are related to the same or a similar topic inside a product. Furthermore, a product

Type	ECLIPSE	MOZILLA	LIBREOFFICE
Number of issues	141,117	751,961	47,542
Number of contributors	2,758	6,962	638
Number of products	229	150	7
Number of components	1,288	1,955	50
Average number of resolved issues per contributor	51.17	108.01	74.52
Median number of resolved issues per contributor	6	3	1
Average number of contributors per component	14.01	33.88	55.62
Average number of words in title	7.96	8.94	9.01
Average number of words in description	87.30	19.64	107.51

Table 8.1.: Key characteristics of our issue datasets.

usually refers to a software sub-project of the community’s project. For example, MOZILLA created three different products for the development of their web browser in BUGZILLA to separate issues related to the different platform versions of their web browser *Firefox*. These products are *Firefox for Android*, *Firefox for iOS*, and *Firefox* (desktop version). Altogether, the datasets were structured as follows. As shown in Table 8.1, the MOZILLA dataset includes 150 products and 1,955 components in which about 34 contributors were active on average. The ECLIPSE dataset consists of 229 products and 1,288 components in which about 14 developers were active on average. The reported issues of the LIBREOFFICE dataset were dispersed among 7 different products and 50 components (about 56 active contributors / developers on average). According to these figures, many developers work in more than one component. For example, if the development activity of all 2,758 ECLIPSE developers would be equally distributed across all 1,288 ECLIPSE components and assuming that every ECLIPSE developer only works in one component, the expected average number of developers per component would be 2.14 instead of 14.01.

Figure 8.2 shows the histograms of the developer activity in the three different open-source communities ECLIPSE (Figure 8.2a), MOZILLA (Figure 8.2b), and LIBREOFFICE (Figure 8.2c). The histograms represent the activity based on our offline datasets from January 2008 until December 2018. Thereby, the developer activity directly reflects the number of developers / contributors (y-axis in logarithmic scale) who resolved a certain amount of issues (x-axis). According to this activity (see also Table 8.1), half of the contributors in ECLIPSE (median) did not resolve more than 6 issues (MOZILLA: 3, LIBREOFFICE: 1) and nearly every fourth contributor (25.5%) only worked on a single issue (MOZILLA: 35.8%, LIBREOFFICE: 54.4%).

8.3.2. Recommendation Approach

The recommendation approach presented in this section describes a content-based recommender system which is based on supervised classification. We trained a classifier with issues resolved by the contributors in order to learn a prediction model. This model was then used to recommend relevant

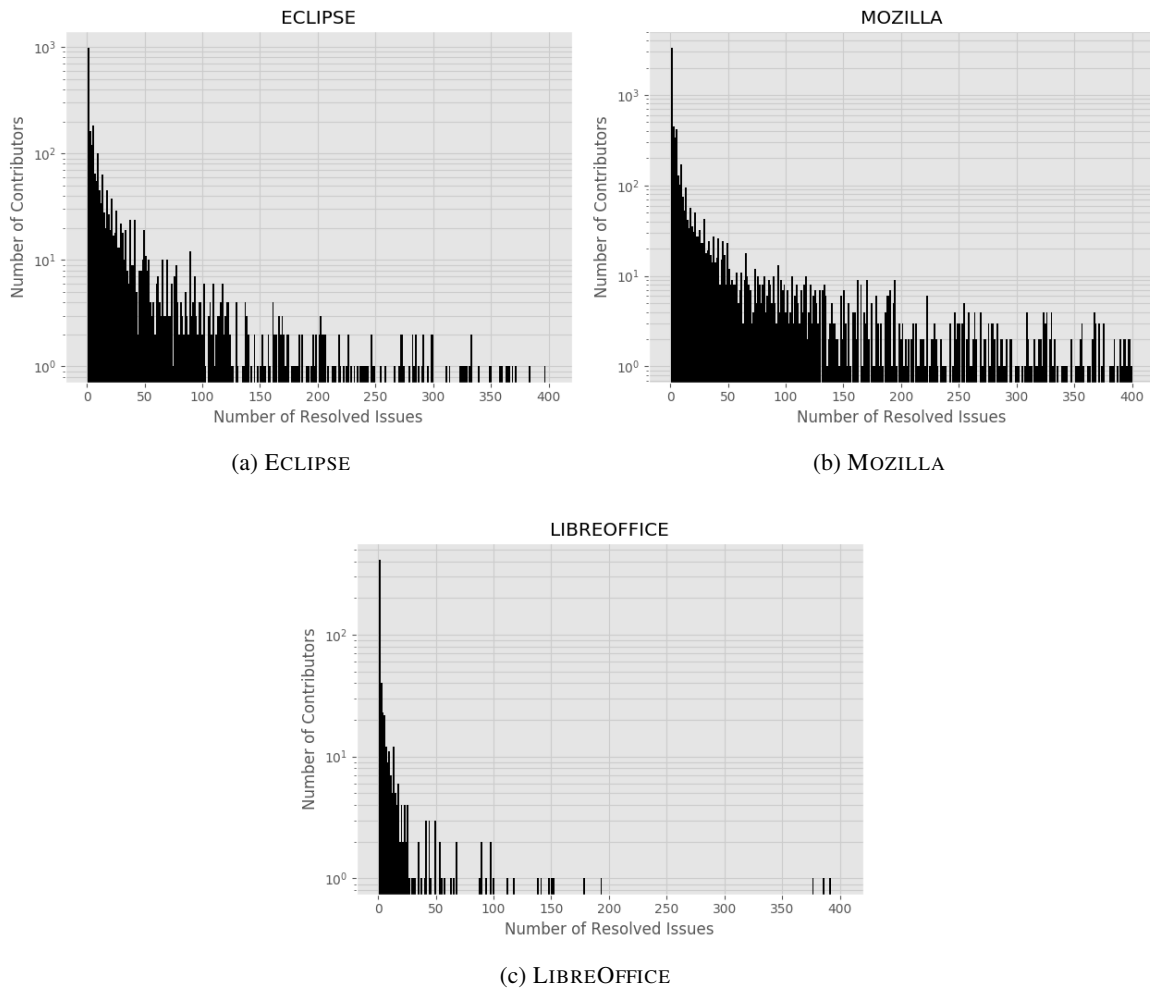


Figure 8.2.: Histograms of the developer activity in different open-source communities (ECLIPSE, MOZILLA, and LIBREOFFICE). The x-axis refers to the number of resolved issues (ascending order) and the y-axis reports the corresponding number of contributors in logarithmic scale.

issues to the developers / contributors. Before training, the (raw) data of the issues had to be converted into a proper format (*feature vectors*) which is suitable for the training of a classifier. The conversion of the (raw) issue data happened in two phases, *text preprocessing* and *feature extraction*.

Text Preprocessing. Following recent research in the field of issue analysis (Stanik et al., 2018; Bhattacharya et al., 2012; Anvik et al., 2006; Anvik and Murphy, 2011; Helming et al., 2010), we focused on the *title* and the *textual description* of the issues and applied several *natural language processing* (NLP) steps, such as *tokenization*, *data cleaning*, *lemmatization*, and *synonym replacement*.

Figure 8.3 provides an overview of the steps performed for preprocessing and feature extraction. We

first lowercased the title and the textual description of every issue and combined them into a single string. After that, we tokenized and split each issue's string into proper linguistic units (*bag of words*) and removed stop words as well as special characters from the text. Furthermore, synonyms were replaced and lemmatization was applied to the remaining tokens by using the *Python Pattern library* (De Smedt and Daelemans, 2012).

The mentioned steps were necessary to reduce the size of the token list and the dimensionality of the vocabulary by combining a variety of inflected or similar terms sharing the same meaning (Korenien et al., 2004). This resulted in a less complex (trained) model which can counteract overfitting scenarios more efficiently.

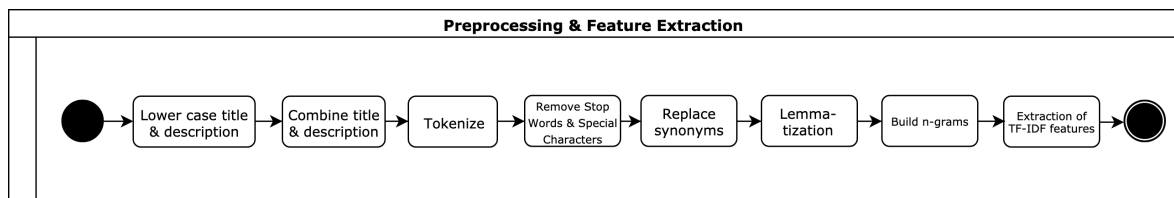


Figure 8.3.: Steps performed for preprocessing and feature extraction.

Feature Extraction. Before feature extraction, we split the set of issues into a training (80%) and a test set (20%). Due to the large number of contributors (see Section 8.3.1), we decided to follow a binary classification approach rather than a multi-class classification approach. Hence, the contributors were not used as labels (i.e., categories) like in standard multi-class classification. Instead, we created a user profile for each developer by considering the tokens of those issues from the training set which were resolved by this developer. Moreover, for each token in the user profile we measured the number of times the token occurred in the issues and preserved this information for further processing. The developers who only occurred in the test set but not in the training set were removed, since we cannot build user profiles for these persons.

Given the user profiles of the developers and our pre-processed issues, we created training and test samples for each contributor. In order to generate the samples for a contributor, we first analyzed in which components in the *training* set (see Section 8.3.1) he or she was active. Whenever there existed issues assigned to a developer in a component (according to the training set), we considered the respective developer as an active contributor in this component. We used all issues of these components from the whole dataset (i.e., training and test set) and generated a contributor-sample for each issue. Each contributor's sample consisted of a feature vector and a label. The sample's feature vector contained the TF-IDF values (*term-frequency inverse document frequency*) of the 30 most frequent words / tokens of the contributor (the *profile tokens*) and the TF-IDF values of the issue's tokens (the *issue tokens*). In addition to these (uni-gram) tokens, we also included n-gram tokens ($n > 1$) of the

profile tokens as well as *n-gram tokens* of the *issue tokens*. Furthermore, we used *grid search* to filter out rarely occurring uni-gram and n-gram tokens (see Section 8.4). The label was either set to TRUE or FALSE, depending on whether the issue was assigned to the developer or not. For example, assuming a developer A was active in the components X and Y according to the training set and active in the components X, Y, and Z according to the test set, we would generate A's samples for every issue which appears in component X or Y. Also, the sample's label would be set to TRUE (i.e., indicating that "developer A resolved the issue corresponding to the sample") if the sample's issue is assigned to developer A, otherwise the label would be set to FALSE (i.e., indicating that "developer A did not resolve the issue corresponding to the sample"). However, even if we create samples for A for every issue which appears in the component X or Y, we do not create any sample for those issues which appear in the component Z (or any other component). This is due to the reason that we only consider the training set in which the contributor was active (in this case, X and Y) for the determination of a contributor's activity.

Classification. The major objective of our content-based recommender is the recommendation of issues that are relevant for a developer / contributor. Each of the developers' samples either uniquely refers to a *single* issue from the training set (TR) or test set (TE). A sample's label can either be set to TRUE or FALSE, expressing whether the issue was assigned to the developer or not (our relevance criterion). Hence, the recommendation task was to train a single binary classifier with the developers' samples covering all issues from TR. The learned model was then used to predict whether a (yet unseen) test sample (representing an issue from TE and a developer) is relevant for the sample's developer or not. As already mentioned before, these test samples cover all test issues of every component as well as all developers who were active in the component according to the training set. We trained binary classifiers based on *Multinomial Naïve Bayes*, *Decision Tree*, and *Random Forest* and compared each classifier with each other (see Section 8.4). The *Multinomial Naïve Bayes* classifier defined our baseline. Our implementation was based on the *Scikit-learn machine learning library*⁴.

As already presented in Figure 8.2 (see Section 8.3.1), most developers are not highly active contributors, which implies that only a small fraction of issues in a component is assigned to the same person. This means that most of a contributor's samples are FALSE samples (i.e., the samples' issues are not assigned to him / her) and only a few samples are TRUE samples (i.e., the samples' issues are assigned to him / her). Hence, the FALSE class dominates in TR as well as in TE. This class imbalance can negatively bias the classification and hinder the classifier from learning the relevant patterns in the data. In order to consider the class imbalance in the training of the classifier, we defined a lower class-weight (0.3) for the FALSE class and a higher class-weight for the TRUE class (0.7) and passed these values as a configuration parameter to the classifier.

⁴Scikit-learn library: <https://scikit-learn.org>

For each classifier, a prediction model was trained with TR and tested with TE. In order to recommend relevant issues to a developer (called A), the recommender system uses the trained prediction model such that all test samples of A are evaluated with the model. The issues of the positively predicted test samples are then recommended to A. Hence, the recommended issue list only includes issues for whose corresponding samples the model predicted TRUE.

ECLIPSE plugin. As part of our work, we also developed a plugin⁵ and a web service to recommend relevant issues to the developers of the ECLIPSE community (see also Section 8.2 and Figure 8.1). Before the web service is deployed on the server, a prediction model is trained with all resolved issues from our ECLIPSE offline dataset. In order to keep our model up to date, we refetch new issues from the ECLIPSE BUGZILLA web API every day and retrain our prediction model with the new issues. The ECLIPSE developers can download our ECLIPSE plugin from the ECLIPSE marketplace and install the plugin inside their local ECLIPSE *integrated development environment* (IDE). Whenever a developer opens the ECLIPSE IDE, the plugin is loaded and our web service fetches *unresolved* issues of the components in which the developer was previously active, from the ECLIPSE BUGZILLA web API. For every fetched issue, our web service generates a new contributor-sample which combines the TF-IDF values of the 30 most frequent words / tokens of the developer as well as the TF-IDF values of all tokens of the issue’s preprocessed title and description (see *Feature Extraction*). Each sample is then individually passed to the trained model which either predicts TRUE or FALSE. The issues of the positively classified samples are then included in the recommendation list. In addition to that, the model also returns a probability / confidence value between 0.0 and 1.0 (values closer to 1.0 indicate more certainty than values closer to 0.5) which is used to rank the recommendation list before it is transferred to the ECLIPSE plugin and shown to the developer.

8.4. Evaluation & Discussion

In order to select the best hyperparameter combination for each classifier and dataset (see Section 8.3.1), we used *k-fold cross validation* ($k = 10$) in combination with *grid search* (Bergstra et al., 2011). For each of our three datasets (ECLIPSE, MOZILLA, LIBREOFFICE), we split the dataset into a training set (80%) and a test set (20%). Furthermore, we divided the complete training set into ten distinct folds of equal size. While nine folds were used for training, the remaining fold was used to validate the trained model. The folds used for training and validation varied in each iteration (in total, there were 10 iterations). The best hyperparameters for each classifier are presented in Table 8.2. The parameter *max_features* refers to the number of used features (n-gram tokens). Depending on this value (*max_features* = m), only the m most frequent n-gram tokens of the training samples were used in the feature vectors of all samples. The parameter *min_df* indicates that n-gram tokens which generally occur very rarely, should be ignored. For instance, the expression *min_df* = 3 means that

⁵ECLIPSE plugin: <https://github.com/OpenReqEU/eclipse-plugin-vogella>

n-gram tokens which occur less often than three times, are ignored. The parameter *ngram_range* states which types of n-grams (e.g., unigram, bigram, etc.) are part of the final feature vector. For example, *ngram_range* = (1, 3) indicates a combination of unigrams, bigrams, and trigrams.

Classifier	ECLIPSE	MOZILLA	LIBREOFFICE
Multinomial Naïve Bayes	max_features=log2 min_df=2 ngram_range=(1,3)	max_features=sqrt min_df=3 ngram_range=(1,2)	max_features=sqrt min_df=3 ngram_range=(1,2)
Decision Tree	max_features=sqrt min_df=2 ngram_range=(1,3)	max_features=log2 min_df=2 ngram_range=(1,2)	max_features=sqrt min_df=3 ngram_range=(1,2)
Random Forest	max_features=log2 min_df=3 ngram_range=(1,4) n_estimators=4000	max_features=log2 min_df=3 ngram_range=(1,2) n_estimators=4000	max_features=sqrt min_df=2 ngram_range=(1,3) n_estimators=3000

Table 8.2.: Hyperparameter optimization using grid search.

Table 8.3 gives an overview of the dataset’s vocabulary size and the number of constructed contributor-samples (data records) used for training and testing. The size of the feature vector (i.e., the number of features) depends on the vocabulary size and is limited by the hyperparameter combination (*max_features*, *min_df*, and *ngram_range*) of the respective classifier which was determined via grid search as already mentioned before (see Table 8.2).

	ECLIPSE	MOZILLA	LIBREOFFICE
Training records	444,078	6,332,667	252,202
Test records	111,019	1,583,167	63,051
Vocabulary Size	230,503	383,540	59,449

Table 8.3.: Overview of the number of constructed data records and the vocabulary size of the different datasets.

It is important to mention that we optimized the hyperparameters to focus more on precision rather than on recall. In other words, we aimed to achieve a high precision rate which indicates that most recommended issues are correct (i.e., relevant for the developers), whereas a low precision would mean that many recommended issues are not relevant for the developers. However, although a high recall would have been beneficial as well, our main focus was on maximizing the precision. The main reason for this decision was that we wanted to make more accurate predictions (high precision) which might come at the cost of a lower recall rate (smaller list of recommended issues where most of them are correct / relevant) instead of overwhelming the developers with a large number of issues (high recall) where many of these issues are not so relevant for the developers (lower precision). Consequently, the precision rate also served as our main measure to compare the classifiers. This way, we could better address the majority of the developers (see Figure 8.2) who are newcomers (i.e.,

developers who only resolved a few issues). Such newcomers can be considered as developers who usually have less experience in the community. Due to this reason these developers should not be overwhelmed with too many non-relevant tasks / issues. This means that for such developers a high precision is more important than a high recall (i.e., less but correct recommendations are more useful for newcomers than a complete set of recommendations which includes all correct recommendations but also many incorrect recommendations). By considering this aspect, we could find the best hyperparameter combination of a given set of predefined hyperparameters for every classifier by using grid search (see Table 8.2). The best hyperparameter combinations were then used for the final training of the classifiers where we trained the classifiers with the complete training set (i.e., all ten folds of the training set). The trained models were then used to evaluate the prediction quality of the classifiers with the samples from the test set, which remained unseen until this step. The prediction quality of the classifiers is expressed in terms of *precision*, *recall*, and *f1-score* (Davis and Goadrich, 2006).

We used two different baselines to compare the performance of our classifiers: *Constant TRUE predictor* and *Multinomial Naïve Bayes*. A *Constant TRUE predictor* is the best random predictor and represents a dummy-classifier which does not learn anything from the data and follows the simple rule by always predicting TRUE. We compared the results of our classifiers with this simple baseline because only if the results outperform this baseline, the predictions of the recommendation model can be considered as to be obtained from learned patterns inside the data rather than from coincidence. In addition to the simple baseline of the *Constant TRUE predictor*, we chose *Multinomial Naïve Bayes* as our second baseline. The basis of this decision was that this approach is often used as a baseline in text classification tasks (see, for instance, Wang and Manning, 2012) and its underlying classification approach follows basic probabilistic assumptions which are known to work well in practice for (quite) uncorrelated features. This is due to the reason that the Naïve Bayes approach only considers the relationship between single words (or n-grams) and the sample's class, but does not take into account the interdependencies between the non-adjacent words (or non-adjacent n-grams) which may correlate. Especially, the correlation between a contributor's profile tokens and the issue tokens (represented by the same contributor-sample, see Section 8.3.2) is crucial for a classifier to predict whether the issue is relevant for the contributor or not. In contrast to *Multinomial Naïve Bayes*, both other algorithms (*Decision Tree* and *Random Forest*) used in our evaluation allow to detect such interdependencies. This is important because the relationship / correlation between the words / n-grams represents valuable learning information from which a classifier can benefit.

Figures 8.4, 8.5, and 8.6 present the evaluation results for all three datasets. The evaluation results have been separated into different figures according to the used evaluation metrics *precision* (Figure 8.4), *recall* (Figure 8.5), and *f1-score* (Figure 8.6). In addition to these plots, the numeric results were attached to this thesis and can be found in the appendix of this thesis (see Tables B.1, B.2, and B.3 in Appendix B). The comparison of the results presented in Figure 8.4 and 8.6 shows that

all algorithms (including our *Multinomial Naïve Bayes* baseline) could outperform the (simple) baseline of a *constant TRUE predictor*, in terms of *precision* and *f1-score*. Regarding the *f1-score* metric, the aforementioned constant predictor is the best *random predictor* with a recall rate of 1.0 (see Figure 8.5) and a precision (see Figure 8.4) depending on the rate of positive samples available in the test set. For example, the ECLIPSE dataset contained 10.06% positive / TRUE test samples (MOZILLA: 7.42%, LIBREOFFICE: 9.23%). Hence, the expected precision rate of a *constant TRUE classifier* in our experiment for the ECLIPSE dataset would be around 0.10 (MOZILLA: 0.07 and LIBREOFFICE: 0.09) and the expected *f1-score* considering the aforementioned recall of 1.0 would be around 0.18 (MOZILLA: 0.13 and LIBREOFFICE: 0.17). Our results show that all three classifiers (*Multinomial Naïve Bayes*, *Decision Tree*, *Random Forest*) could surpass this simple baseline. This allows us to make the conclusion that all classifiers could learn relevant details / patterns from the data in order to build a classification model which is suitable for making reliable predictions.

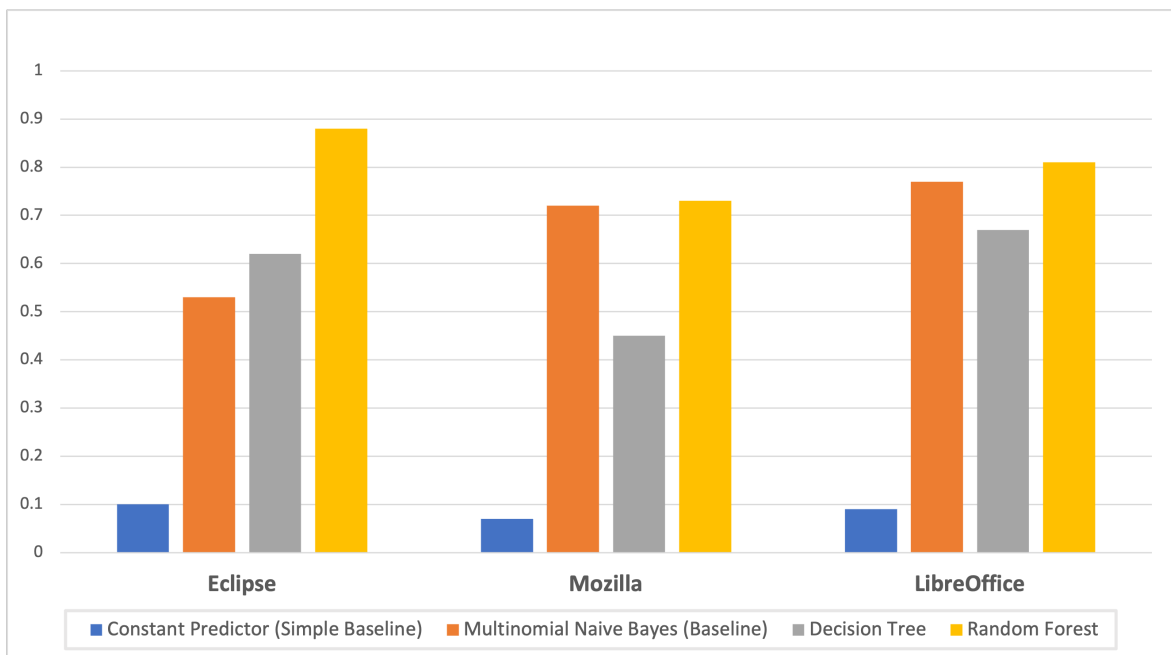


Figure 8.4.: Precision scores of the different classifiers and the constant predictor baseline. The highest precision rates for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE) were achieved by Random Forest.

ECLIPSE dataset. In the ECLIPSE dataset, the highest *precision* was achieved by Random Forest at 88%. Furthermore, Random Forest could also outperform our Multinomial Naïve Bayes baseline as well as the Decision Tree classifier in terms of *recall* and *f1-score*. The hyperparameters which achieved this result were: $n_estimators = 4000$, $min_df = 3$, $n_gram_range = (1,4)$ and $max_features = \log 2$. The Decision Tree classifier also achieved acceptable results in our experi-

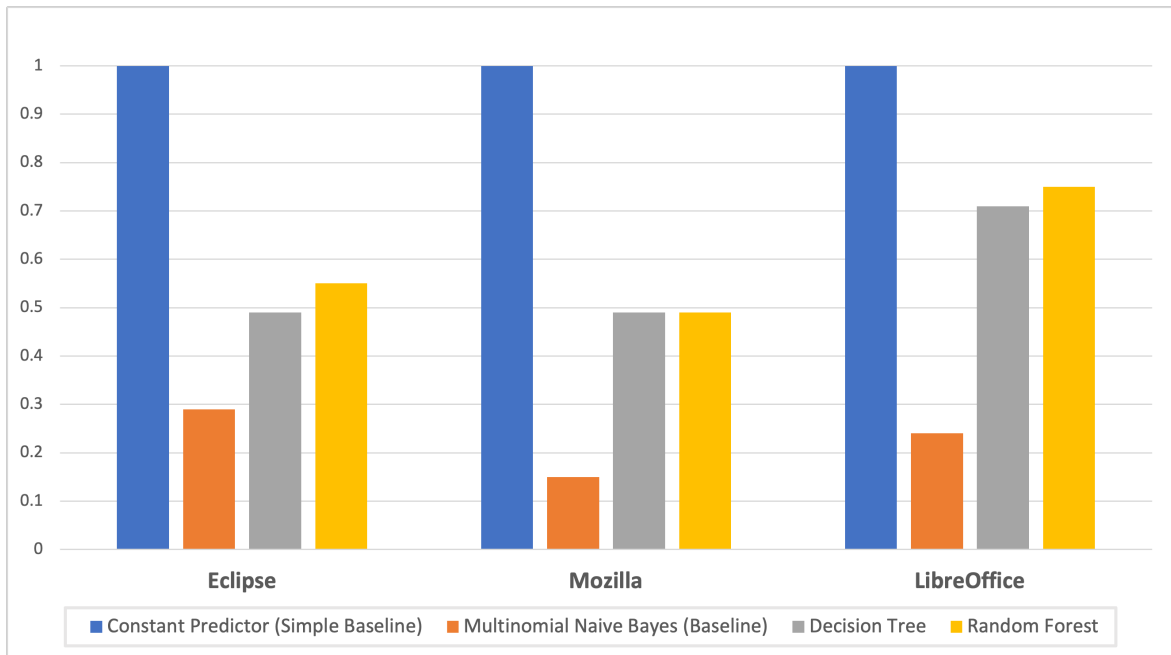


Figure 8.5.: Recall rates of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE). The constant predictor baseline represents a very simple ”dummy” classifier that always recommends *all* available issues and therefore achieves the highest possible recall rate.

ment (*precision*: 0.62, *recall*: 0.49, and *f1-score*: 0.55). Even though Decision Tree achieved a recall of 49% which is similar to the recall of Random Forest, its precision rate is clearly behind the results of Random Forest.

MOZILLA dataset. The hyperparameters of the classification model based on Random Forest that achieved the best result (in terms of all three measures: *precision*, *recall*, and *f1-score*) were: $n_estimators = 4000$, $min_df = 3$, $n_gram_range = (1, 2)$ and $max_features = \log 2$. In terms of *precision*, the Random Forest classifier was closely followed by our (second) baseline predictor (Multinomial Naïve Bayes classifier), which achieved a high precision of 72% but a very low recall rate of 0.15. Even though the *precision* of the Decision Tree classifier could not outperform the *precision* result of the baseline predictor for this dataset, its recall rate of 0.49 clearly surpasses the baseline which is identical with the recall rate attained by Random Forest.

LIBREOFFICE dataset. The highest *precision* achieved with this dataset was 81% using Random Forest. Moreover, Random Forest also attained the highest recall rate of 75% and the highest *f1-score* value of 0.78 for the LIBREOFFICE dataset. The best hyperparameter combination for Random Forest was: $n_estimators = 3000$, $min_df = 2$, $n_gram_range = (1, 3)$ and $max_features = \sqrt{}$. On the contrary, the Multinomial Naïve Bayes classifier achieved a similar precision (4% less) much like our

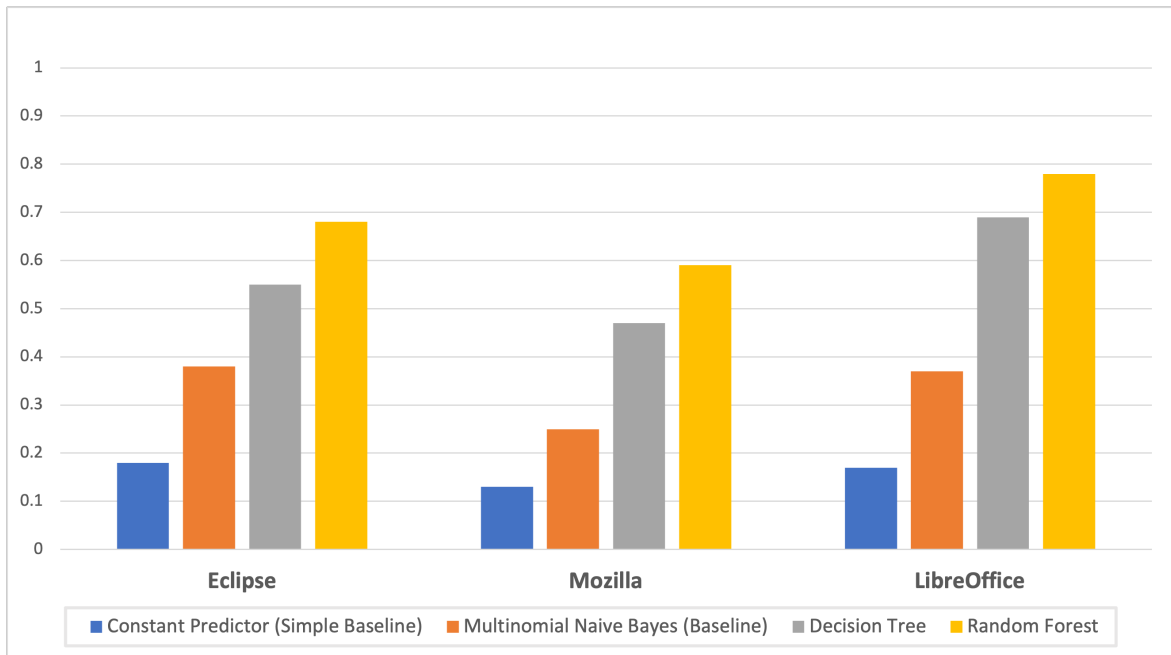


Figure 8.6.: F1-scores of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE).

Random Forest classifier, but was not able to attain a high recall rate (only 24%). The Decision Tree classifier achieved a recall of 71% (which is comparable with the Random Forest classifier's recall), but only a precision of 67%.

Summary. To summarize, the presented evaluation results are quite promising. In particular, considering the difficulty of this recommendation task and our optimization towards *precision* (instead of *recall*), the classifier based on the Random Forest algorithm achieved good results in terms of *precision* (see Figure 8.4) and acceptable *recall* rates (see Figure 8.5) on all three datasets. Random Forest achieved its highest precision (88%) on the ECLIPSE dataset which clearly outperforms all other classifiers. Given the following three facts, the difficulty to learn a prediction model for LIBREOFFICE can be considered as being the highest of all three datasets: (1) the LIBREOFFICE dataset was significantly smaller when compared with the other two datasets (in terms of issues, contributors, and components; see Table 8.1), (2) the median number of resolved issues per contributor was the lowest of all three datasets, and (3) the average number of contributors per component was the highest. Thus, the results achieved with Random Forest on the LIBREOFFICE dataset are even more remarkable than the results of the other two datasets.

Even though these results indicate a first step in the right direction, one limitation of our approach is that the trained models cannot recognize some developers working in different components. This is

due to the reason that we generate a different user profile (set of *profile tokens*) for the same developer in every component where he / she was active. This means that a contributor-sample always contains only profile tokens of issues (resolved by the contributor) from *one* component even though the contributor works in multiple components (according to the training set; see Section 8.3). By using unique user profiles for the same developer across different components, the set of profile tokens can be further enriched with useful tokens / keywords extracted from issues in other components. This improves the quality of the profile tokens as well as the quality of the contributor-samples (which contain the profile tokens). Moreover, this might also have a positive impact on the performance of the prediction models as the classifier is fed with more / better keywords and can learn connections between issues resolved by the same developer across different components (due to the unique set of profile tokens).

Although all results were achieved with the set of hyperparameters listed in Table 8.2, a different combination of these hyperparameters can lead to (slightly) different results. The hyperparameters can be adjusted according to different needs (for example, in order to increase the *recall*) by either modifying the hyperparameters manually or by using other techniques such as *Random Search* to further optimize the models (Bergstra and Bengio, 2012). For certain applications (e.g., when expert users want to review the complete set of issues relevant for them) a higher recall achieved with different hyperparameters might be more important than a high precision. The main challenge of this recommendation task was to correctly predict and recommend those issues of a component to a developer / contributor which were actually assigned to him / her given a possible choice of about 14 active contributors on average per component in ECLIPSE (MOZILLA: 34 and LIBREOFFICE: 56; see Section 8.3.1). Another hurdle for the classifiers was that most developers were only assigned to a very few issues, which negatively influences the quality of their user profiles. Considering these challenges, our preliminary results of the presented work represent a good orientation towards future work.

8.5. Future Work

Within the scope of future work, we plan to evaluate our issue recommendation approach on the basis of further open-source projects. Examples thereof are the GNU GCC project (GNU Compiler Collection), the LINUX KERNEL project, the GNOME project, the KDE project and the QT project. A known limitation of the existing solution is that our recommendation model follows a content-based approach which can only recommend issues to a developer that are similar to issues the developer resolved in the past. In particular, the model lacks the ability to make smarter recommendations considering serendipity aspects across different components of the same product. As shown in Figure 8.1, we allow the users of our ECLIPSE plugin (i.e., the ECLIPSE developers) to provide feedback in terms of issue likes, issue dislikes, or issue snoozes ("*remind me later about this bug*"). At the moment, we use our users' feedback to hide certain issues (in case of dislike) and to re-rank (in case of like) the

recommendation list. In the future, we plan to use this data to automatically improve and adapt our recommendation model in order to foster serendipity and counteract the aforementioned limitation of our current (content-based) approach. In order to reach a much broader community of developers, we also want to address developers who are using different integrated development environments (IDEs) such as the very popular products from the software company JETBRAINS⁶. Therefore, we intend to develop an open-source plugin for some of JetBrains' IDEs, such as IntelliJ IDEA, CLion, etc. In addition to that, we will also provide a web UI as an alternative solution. Furthermore, we will focus on improving our recommendation support for *ramp-up* (i.e., *cold-start*) scenarios. In particular, we intend to increase the retention rate by recommending requirements to new developers. Finally, another idea for future work is to provide the currently developed recommendation functionalities in a similar fashion for GITHUB.

8.6. Conclusion

We introduced a novel content-based recommendation approach that supports contributors of open-source communities in identifying the relevant requirements to work on. To demonstrate the potential of the developed approach we implemented a plugin for the commonly used ECLIPSE platform that supports contributors in selecting the next issues from the BUGZILLA platform. The recommendation approach performs much better than comparable approaches in terms of prediction quality (represented by *precision* and *recall*). To underline the potential of the proposed approach we used publicly available datasets of three different open-source projects (ECLIPSE, MOZILLA, and LIBREOFFICE). Since we received very positive feedback from contributors of the ECLIPSE IDE, we regard the continuous extension and improvement of the recommendation approach as a major topic for future work. In addition, we will conduct further usability studies related to the developed plugins.

⁶JETBRAINS: <https://www.jetbrains.com/>

Intelligent Recommendation & Decision Technologies for Community-Driven Requirements Engineering

The contents and results of this chapter are based on the research work published in Samer et al. (2020). The author of this thesis provided major parts of this chapter in terms of literature research, the user studies and wrote major parts of this chapter.

9.1. Abstract

Requirements engineering (RE) represents a critical phase in the management and planning of software projects. One of the main reasons for project failure is missing or incomplete RE. In order to reduce the risk of project failure, there exists a high and urgent demand for applying intelligent technologies in RE. Since the RE process is mainly decision- and community-driven, *recommender systems* are supposed to be applied in this particular context to support stakeholders in decision-making and, hence, to increase the quality of the decisions taken by the stakeholders. This chapter introduces a variety of innovative recommendation tools developed within the scope of the European Horizon 2020 research project OPENREQ. Moreover, we give an overview of user studies conducted to evaluate our approaches and present final results of selected studies. The study results indicate that the developed concepts have the potential to significantly improve the quality of requirements definition and requirements prioritization.

9.2. Introduction

Requirements engineering (RE) plays an important role in software development. In general, RE represents a branch of systems engineering which deals with the definition and fulfilment / implemen-

tation of desired properties and constraints of software-intensive systems. The major phases of the RE process are the *elicitation and definition of requirements*, the *negotiation of requirements*, *quality assurance*, and *release planning*. RE can be considered as a critical phase in a software project since poor (or missing) RE can (1) cause a software project to miss important deadlines (due to a late discovery of serious issues in the RE model), (2) lead to increased costs which can exceed the project budget, or (3) even result in project failure (Davis, 2005; Leffingwell, 1997; Mobasher and Cleland-Huang, 2011). In fact, research shows that the follow-up costs can add up to 40% of the overall project costs (Davis, 2005; Leffingwell, 1997). In the worst case, the project might miss important deadlines or even fail. Consequently, RE constitutes a high risk factor for the success of a project. Hence, there is a high demand for applying intelligent technologies to support stakeholders in RE, in order to mitigate these project risks. In particular, *recommender systems* (RS) are predestined to support stakeholders in different decision-making scenarios which represent the core foundation of the RE process (Felfernig et al., 2013; Mobasher and Cleland-Huang, 2011). RS can support stakeholders in RE tasks, such as *requirements definition*, *release decisions*, *stakeholder identification*, and *dependency detection* (Mobasher and Cleland-Huang, 2011; Ninaus et al., 2014). Beyond the use of RS in RE, further intelligent and automated solutions based on *artificial intelligence* can be used to support stakeholders in RE.

This chapter presents innovative applications of intelligent recommendation and decision technologies in RE which are based on artificial intelligence. These technologies were developed within the scope of the European research project OPENREQ¹. The major aim of OPENREQ is to address the aforementioned issues by providing an innovative and intelligent tool support, which might change the way RE stakeholders think about and work with requirements. Following the objective to foster high quality decision-making, OPENREQ offers intelligent solutions for all phases of the RE process. OPENREQ has even the potential to update current software engineering methodologies and introduce new roles in software organizations. For instance, with OPENREQ the boundaries between marketing, RE, and maintenance should be reconsidered. The outcomes of the OPENREQ project touch a broad set of different communities. OPENREQ provides actionable feedback for novel contributions, software practitioners for the scientific community as well as solid foundations for the open-source community. The work presented in this chapter deals with the analysis and extension of current methodologies on (1) stakeholder and user involvement in a software life-cycle, and (2) distributed requirements engineering and management. We provide a framework for processes and methodologies that support stakeholders in using the OPENREQ platform to achieve high efficiency and quality in requirements elicitation and management. The focus lies on extending agile, reuse-driven methodologies, community-centred participative product development, and open innovation methodologies. The core contributions of the developed recommendation tool suite presented in this chapter, provide genuine added value for traditional software development institutions and open-source communities

¹OPENREQ: <https://www.openreq.eu>

in terms of (1) more efficient information exchange among stakeholders, (2) increased RE quality by providing quality-related feedback and advanced conflict-resolution, and (3) reduced risk of project failure by providing immediate feedback to requirements engineers early in the process.

The remainder of this chapter is structured as follows. Section 9.3 provides an overview of the developed OPENREQ tool suite covering a broad set of different intelligent recommendation technologies. In Section 9.4, we show the user interface of the RE platform OPENREQ!LIVE which provides a central platform for the use of the recommendation tools. Section 9.5 presents the design and results of several user studies that evaluate the different recommendation approaches. Section 9.6 outlines related work and provides some ideas regarding future work for recommender systems in the field of RE. Finally, the chapter is concluded with Section 9.7.

9.3. OPENREQ Recommendation Technologies

A broad collection of different recommendation tools has been developed within the scope of the research project OPENREQ. Thereby, we focused on the techniques which are of the highest relevance for the RE process. These recommendation tools are based on common recommendation concepts, such as *content-based filtering* (Pazzani and Billsus, 2007), *collaborative filtering* (Ekstrand et al., 2011; Goldberg et al., 1992), *knowledge-based recommendation* (Felfernig et al., 2014), and *group recommendation approaches* (Masthoff, 2015; Felfernig et al., 2018). The objective of the developed recommendation approaches is to improve the efficiency and the quality in requirements elicitation and management. Our approaches are supposed to support stakeholders working on different RE-related tasks, such as the *assignment of stakeholders to requirements*, the *elicitation of requirements*, the *identification of requirement dependencies*, and the *prioritization of requirements*.

The developed techniques and tools follow the community-driven OPENREQ approach for modern software RE. Figure 9.1 presents the general flow of the OPENREQ approach and demonstrates on a basic level how the participants (users, communities, and stakeholders) interact with the RE process. As shown in the figure, the basic idea of the OPENREQ approach is that users and communities provide valuable feedback (implicit and explicit) which can be analyzed and used as learning input for OPENREQ's intelligent technologies. All stakeholders (requirements engineers, developers, and users) provide expertise and define preferences which are considered as input for the RE process. Beyond this, OPENREQ aims to use knowledge from previous / past projects and the history of current projects as input to further optimize decision support provided by the developed recommendation and decision support tools. An overview of these tools is given in the remainder of this section. All tools were developed as standalone open-source web services².

²Source code is published on GITHUB: <https://github.com/OpenReqEU>

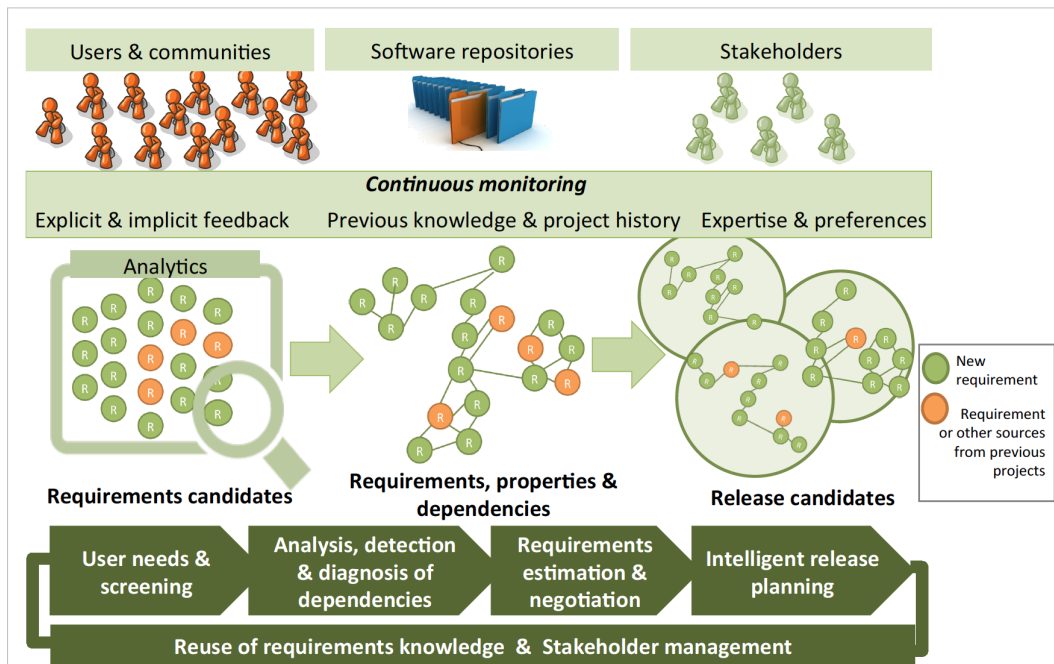


Figure 9.1.: Overview of OPENREQ's requirements engineering approach.

9.3.1. Requirements Elicitation

The elicitation of requirements represents a task in the initial phase of the RE process, where requirements of a software project are jointly defined and collected by the project stakeholders. The traditional way is that the stakeholders provide the fundamental elements for the definition of a requirement which can be textual descriptions, scenario descriptions, use cases, or mock-up illustrations of prototypical user interfaces. Based on these elements, stakeholders select the relevant requirements. However, the process of distinguishing between the elements which define a new requirement and those elements that further explain or describe an already defined requirement, is often very time-consuming for stakeholders. To support this task, OPENREQ provides a classification service called *OPENREQ Classification Service* (Falkner et al., 2019) which is based on supervised machine learning. It focuses on the classification of textual descriptions and helps to determine whether a piece of text (paragraph) as a part of a formatted text document (Microsoft Word) defines a new requirement (denoted as REQ) or represents a description (denoted as PROSE) which is related to a previously defined requirement. Paragraphs classified as PROSE are automatically linked to the corresponding requirement. The hierarchy of classified requirements and PROSE paragraphs is then converted into a format suitable for the requirements management tool IBM DOORS³. Misclassified samples have to be manually corrected by requirements managers.

³IBM DOORS: <https://www.ibm.com/us-en/marketplace/requirements-management>

Another approach supporting the elicitation of requirements has been developed by Stanik et al. (2019) (see also Stanik and Maalej, 2019). The *Orchestration Service* of OPENREQ applies this approach in order to intelligently extract requirements from social media channels. The basic idea of the approach is to bridge the gap between requirements engineers and customers in an agile development process that relies on continuous feedback from the customer. The service opens new feedback channels for customers to report issues and ideas regarding the developed software products. The service intelligently analyzes and classifies all messages (tweets) which appear in these channels by using supervised machine learning (Stanik and Maalej, 2019). The classification model can either be trained by using traditional machine learning or deep learning. Messages can be of any kind of requirement such as a *feature request* (new feature which should be included in the next release), a *bug report* (a bug which leads to malfunction of the software), or an *inquiry* (a question related to different aspects such as usability, compatibility, or questions on how to use the software). All other messages which do not represent (or do not refer to) a requirement are classified as *irrelevant*. All tweets are preprocessed and cleaned in order to avoid obvious spam messages (i.e., *irrelevant*) at an early stage of the classification process. Further (less obvious) spam messages or any other kind of unclear messages are detected by the prediction model and automatically classified as *irrelevant*. The requirements engineers are responsible to review the list of recommended requirements (messages classified as requirements) and select the final candidates which are then included in the requirements model of the software product.

Most software projects contain reusable parts of the functionality (e.g., user authentication systems used in online applications) which often represent core components of other software projects as well. Consequently, the requirements related to the implementation of such reusable components can be reused in new software projects. Recommender systems aim to support stakeholders in the definition of new requirements by suggesting requirements which are related to the content of already defined requirements (Ninaus et al., 2014). The presentation of reusable requirements (extracted from other software projects that are related to the current project) represents a large potential for recommender systems during requirements elicitation. OPENREQ provides a recommendation component called *Similar-related Requirements Recommendation Service* which is a web service that addresses this aim. The main goal of the service is (1) to foster the systematic reuse of requirements such that the efficiency of the RE process can be improved, and (2) to analyze the requirements of a project in order to detect duplicates and related requirements. The service recommends requirements that are similar or related to a given set of requirements by intelligently analyzing the given requirement set of the current project (*inter-project analysis*) as well as requirements of different existing software projects (*cross-project analysis*). The service compares the semantics of the words from the description of the requirements to find and recommend closely related requirements. An additional part of the *cross-project analysis* is the recommendation of requirements of reusable components from other projects. Typical software projects usually have a large assortment of requirements (more than 100 or 1,000 requirements) which makes the elicitation of requirements to become one of the most important and

time-consuming tasks in RE. OPENREQ tackles the aforementioned issues and allows stakeholders to save much time and to reduce the risk of overseeing important requirements in a project. It is important to mention that incomplete or faulty RE (e.g., caused by the late discovery of important requirements) can have a negative impact on software quality (Boehm, 1981; Davis, 2005; Cleland-Huang et al., 2003). This underlines the need for using recommender systems in this critical phase of the RE process.

9.3.2. Requirement Dependency Detection

The identification of dependencies between requirements represents another important task in RE. In this task, requirements are typically analyzed pairwise, in order to find all dependent requirement pairs. There exist different types of dependencies / relationships (e.g., *requires*, *includes*, *excludes*, or *similar* (Samer et al., 2019; Felfernig et al., 2018)) expressing different meanings. The early and complete discovery of all dependencies has a significant impact on the successful completion of a software project (Li et al., 2012). Incomplete, inconsistent, or incorrect dependencies can induce serious consequences in a project, such as increased costs which may exceed project budget, lead to missed project deadlines, or even project failure (Samer et al., 2019; Ruhe, 2010). As already mentioned before, software projects often have a large number of requirements which must be fulfilled in order to successfully complete the project. The number of requirement pairs k to be analyzed for dependencies is a function of quadratic order of the number of requirements n (more formally, $k = \binom{n}{2}$). Hence, the manual analysis of k pairs quickly turns out to be a herculean task / effort for (human) stakeholders. Moreover, with an increasing number of requirements in a project, the probability is very high that a manually defined set of dependencies is incomplete and contains inconsistencies. The complete awareness of all (correct) dependencies directly affects the *release planning* of a software product since the dependencies express relevant information about the compatibility between the requirements as well as the chronological time order in which the requirements should be implemented. An extensive consideration of all dependencies fosters a more fruitful release planning which helps to avoid effortful re-designs and re-implementations later in the project. Consequently, the application of automated technologies which assist the stakeholders in finding requirement dependencies is essential for a software project.

In order to address this issue, two different recommendation approaches have been implemented within the scope of OPENREQ. The first approach is represented by OPENREQ's *Dependency Recommender Service* which analyzes the requirements set of a software project to find dependent requirement pairs. The underlying approach uses *Latent Semantic Analysis* (Deerwester et al., 1990; Landauer et al., 1998) to transform the textual descriptions of the requirements into a low dimensional semantic-space representation (for more details, see Samer et al., 2019). This way, noisy words can be filtered out and more emphasis is placed on semantically-related requirements. The pairs of closely semantically-related requirements are considered as dependent requirements. The main advantage of

this service is that it does not require any labeled training data and can be used once all requirements have been defined in a project. This is due to the reason that the underlying approach uses unsupervised learning (soft-clustering). Moreover, fine-tuned parameters (e.g., the minimal similarity distance between two requirements) avoid that two too closely related (i.e., similar or duplicate) requirements are falsely classified as a dependent pair.

Besides this unsupervised approach, we have developed another approach that is based on supervised machine learning and requires labeled dependency data in order to learn a prediction model (Samer et al., 2019; Atas et al., 2018). This approach goes beyond the level of similarity-based recommendation, and uses probabilistic features which take statistical aspects from the area of *information theory* into account. In contrast to the previous approach, the main benefit of this approach is a significantly increased prediction quality (see results in Samer et al., 2019).

9.3.3. Prioritization and Evaluation of Requirements

A correct prioritization and allocation of all resources and requirements is the fundamental basis for a smooth and efficient schedule in every software project. This involves the evaluation and prioritization of a project's requirements, the assignment of suitable stakeholders to requirements (see Section 9.3.4), and release planning (see Section 9.3.5). In particular, an efficient support of prioritization decisions is essential for a software project. This is due to the reason that a manual prioritization of a large number of requirements is a very time-consuming and effortful process (Xuan et al., 2012). However, the prioritization of requirements is often performed by a single person or a very small group of stakeholders (e.g., by requirements / project managers).

According to research in the field of group recommender systems, more information exchange between decision-makers as well as more people involved in a decision can significantly increase the probability of better prioritizations (Schulz-Hardt et al., 2006; Stettinger et al., 2015). Following these scientific empirical observations, we developed new approaches for group recommendation user interfaces (Samer et al., 2020) which (1) trigger more stakeholder engagement and (2) foster information exchange between the stakeholders. The approaches are based on the concept of *multi-attribute utility theory* (MAUT) (Felfernig et al., 2018; Ninaus et al., 2014) which represents a multidimensional rating scheme. MAUT for groups extends the basic utility-based recommendation for single users to multi-user scenarios where the preferences of the individual stakeholders are aggregated into a recommendation such that the whole group is satisfied with the recommendation. In our approach, we use three interest dimensions (*profit*, *effort*, *risk*) which must be evaluated by every stakeholder for every requirement individually. The preferences of a single user are the user's ratings of all specified dimensions. Strongly diverging ratings of different group members within the same dimension indicate a strong disagreement and are considered as a conflict by the group recommender. The group recommender automatically detects all of such conflicts and presents them to the group members who

are involved in the conflicts. For each conflict of a requirement, the group members are required to discuss the conflict and (after the discussion) to reevaluate the requirement's dimensions which are affected by the conflict. This helps to trigger more communication between stakeholders which positively influences the quality of the requirement prioritizations. Once all requirements have been evaluated and all conflicts have been resolved by the stakeholders, a utility / priority value is determined for every requirement. Based on this priority value, an ordered list of requirements is presented to the stakeholders for further action (e.g., release planning). In sharp contrast to traditional group recommendation approaches which do not take into account the aspects of *delegate voting (liquid democracy)* (Johann and Maalej, 2015), our approach aims to make voting processes more flexible by allowing to transfer voting rights to stakeholders who are the experts with respect to specific requirements and interest dimensions. In other words, the approach harnesses the stakeholders' knowledge and its algorithm allows them to prioritize and delegate at scale.

Another OPENREQ service which supports the prioritization and evaluation of requirements is the *Social Popularity Indication Service*. It provides further relevant input for the evaluation of the requirements by estimating the relevance of a requirement given its overall sentiment and popularity in social media networks⁴. The tool automatically extracts messages from *Twitter* which are related to the textual content of a requirement. The cleaned messages are then further analyzed considering the sentiment, in order to obtain a relevance score for every requirement. This relevance score is updated every day and refers to the social popularity which serves as an indication on how relevant / popular a requirement is for potential users / markets that a software company may want to address.

9.3.4. Stakeholder Recommendation

The task of assigning suitable stakeholders to requirements is essential for the success of a software project (Lim et al., 2010). A complete and correct assignment in the early phases of the RE process is indispensable. With an increasing number of requirements, this task can become very challenging for requirements managers. Stakeholder recommendations can assist requirements managers in the identification of suitable persons who are capable of implementing the requirement or providing a more detailed analysis and description of the requirement. Inspired by existing research (Lim et al., 2010), OPENREQ comes up with two new content-based recommendation approaches which have been implemented as different services.

As described by Samer et al. (2018) (see also Palomares et al., 2018), OPENREQ's *Stakeholder Recommendation Service* implements the first approach. In contrast to traditional stakeholder assignment where requirements managers decide on who is responsible for a requirement, the basic idea followed by this approach is to involve more stakeholders in the assignment decision process. This includes human and artificial stakeholders. Content-based recommenders act as artificial stakehold-

⁴At the moment, the support for social networks is limited to *Twitter*.

ers and propose suitable stakeholder candidates for each requirement based on learned user profiles from historical data. The (human) stakeholders⁵ can extend the list of already proposed candidates (if necessary) by adding further stakeholders to the candidate list. Moreover, the human and artificial stakeholders are asked to evaluate all proposed candidates individually. Given the complete evaluation of the proposed candidates as input produced by the combined power of the intelligent service and the expert knowledge of the stakeholders, a group recommendation system preselects final candidates to be assigned to the requirement. The requirements manager can then either accept the final candidates proposed by the group recommender (no action is required) or choose an alternative candidate (action is required) in exceptional cases. The main benefit of this group-based evaluation approach represents the potential to significantly reduce the overall involvement of the requirements managers in this particular task and it can also lower the risk of overseeing suitable stakeholders.

Our second approach is implemented by the *Issue Prioritizer Service* and it addresses a slightly different scenario where stakeholders receive a list of recommended requirements based on a user profile (see Samer et al., 2019). Such scenarios typically occur in large open-source projects or in large commercial projects with highly fluctuating groups of employees / developers. Our approach uses content-based filtering and is based on supervised machine learning. The service builds a keyword profile based on "old" requirements / issues resolved by a stakeholder / contributor in the past and aims to find new requirements which are similar in terms of the content. In contrast to standard content-based filtering, we do not use similarity metrics but exploit the potential of machine learning by using large requirement / issue datasets. Based on the keyword profile, relevant requirements matching the stakeholder's interests are recommended using supervised classification techniques. Thereby, our content-based classification approach accepts a single feature vector (consisting of keywords of the current requirement and the keywords of the current stakeholder's keyword profile) as input and uses binary classification to predict whether the given requirement is suitable for the stakeholder or not (for more technical details, we refer to Samer et al., 2019). Since the approach is fully automated and does not require any input from other stakeholders (such as evaluations of proposed candidates or suggestions for candidates), the approach is more suitable to be applied in projects where stakeholders / developers can start to work on a new requirement immediately (without waiting for permission from a requirements manager) which is typically the case in large open-source projects. Moreover, the approach also supports flexible working environments where onboarding of newcomers (i.e., new contributors who want to join the project, but are not yet known by the community) is of high importance for the project (see also Stanik et al., 2018).

⁵The group of stakeholders also includes the requirements manager and expert stakeholders working in the domain related to the requirement.

9.3.5. Release Planning and Configuration

Release planning deals with the assignment of requirements to releases. It usually follows the principle of *requirements triage* which is accounted for by the fact that there is only a limited amount of available resources in a project (Ninaus et al., 2014). According to *requirements triage*, requirements are classified into (1) requirements which should not be assigned to any release, (2) requirements which have to be included in an early (or the next) release, and (3) requirements whose assignment to an early release is optional. On an abstract level, this procedure can be regarded as a configuration problem. Raatikainen et al. (2018) and Felfernig et al. (2018) present a recommendation approach in terms of a configuration system which has been developed within the scope of OPENREQ. In their prototype service (named *Release Planning and Consistency Check Service*), the aforementioned requirements triage settings (the three types) are modelled as constraints and serve as main input for the service. Further constraints are dependencies defined between the requirements (see Section 9.3.2), the release dates / deadlines, and the maximum time capacity of each release which limits the number of requirements that can be assigned to the release (based on the specified time effort of the requirement). In addition to these constraints, the complete sets of requirements and releases are used as additional input for the service. Based on the input, the service uses knowledge-based configuration techniques to automatically generate and suggest a list of different release plan candidates which satisfy all defined constraints and represent possible / feasible release plan solutions. The requirements manager can review the recommended list and select one solution as final release plan.

9.3.6. Quality Assurance

Quality assurance deals with the aspect-oriented evaluation of requirements. The aspects which should be evaluated, represent qualitative attributes such as *feasibility* (economic vs. technical feasibility), *consistency* (no requirement must conflict with another), *completeness* (the requirements model must include all necessary requirements), *understandability* (readability / understandability quality of the requirement's description), and *reusability* (for future software projects) (Felfernig et al., 2013). Quality assurance in RE represents the backbone of the RE process and is a highly important measure for preventing mistakes and defects in the development of software products. To that end, our OPENREQ team has developed two services which facilitate and foster quality assurance in RE.

The OPENREQ *Orchestration Service* (introduced in Section 9.3.1), also provides statistics of collected community data from social media channels. More precisely, the service presents general statistics of the analyzed tweets / messages and can automatically keep track of recent activities and changes in the social media channels which are linked to a software project. The statistics visualized by the tool serve as a fundamental means for requirements managers to identify, analyse, and understand the users' desires and the reported problems the users face while using the software product.

Since requirements are usually described using natural language, the textual descriptions of the re-

quirements can often become inherently ambiguous. The serious consequences caused by such ambiguities are often misleading information, inconsistent descriptions, or poor understandability of the requirements. This can further lead to fatal mistakes or defects in the development phase. Common approaches to detect ambiguities are often rule-based and follow certain guidelines. The *Improving Requirements Quality Service* is based on this idea and assists stakeholders in improving ambiguous and unclear definitions of their requirements. Besides pure ambiguity detection, the service provides detailed explanations for each detected ambiguity to the user in terms of basic graphical visualizations. These visualizations represent indications that mark requirements that "smell" which means that the stakeholders should look again at the textual formulations (e.g., if there are some ambiguities introduced by natural language). Further research will be needed to fine-tune the introduced concepts. The ambiguity analysis of the service focuses on syntactic and semantic issues in the wording and phrasing of the requirements. This way, the service helps to improve the quality of a project's requirements model. This avoids serious consequences (mentioned above) which can cause increased time effort and follow-up costs.

9.4. OPENREQ User Interface

In order to provide stakeholders a convenient and user-friendly central access to OPENREQ's recommendation tool suite presented in Section 9.3, we developed a web-based RE platform called OPENREQ!LIVE. OPENREQ!LIVE⁶ is a free collaborative RE platform which fosters the cross-fertilization of ideas between stakeholders and allows them to jointly manage their projects and take full advantage of the recommendation power provided by OPENREQ's tool suite. OPENREQ!LIVE is capable of tackling all of the everyday RE tasks stakeholders face in their software projects. From a technical viewpoint, the platform represents a central hub which combines the functionality of the most relevant OPENREQ services. The platform provides all necessary functionality to create and update the requirements model of a project which includes requirements, releases, dependencies, and further release-related constraints (such as the maximum requirement capacity defined for a release or the release date / deadline) of a project. Moreover, on a project's main page, the platform presents a compact overview of the project which illustrates the defined project structure at a glance (see Figure 9.2). The users of the platform can modify and update the requirements and releases directly on this page.

The evaluation of the requirements defines the basis for the utility-based prioritization discussed in Section 9.3.3. Figure 9.3 shows an example of an argumentation-based MAUT rating interface. Stakeholders are asked to provide textual feedback (in terms of arguments) for every requirement. The stakeholders argue on issues related to the requirement and manually classify the sentiment of their arguments (*positive* (PRO), *neutral* (NEUTRAL), or *negative* (CON)). Moreover, based on the type of issue, every argument has to be assigned to at least one interest dimension $\{profit, effort, risk\}$ (see

⁶OPENREQ!LIVE: <https://github.com/OpenReqEU/openreq-live>

ID	Title	Description	Status	Utility
44	PDF Generation	Export project information to pdf (https://mast-tuleap.informatik.uni-hamburg.de/plugins/tracker)	New	5.25 5 user voted
48	Requirement Attachments	Add and modify attachments of a requirement; Users can only upload (i.e., add) attachments and	New	3.80 4 users voted
27	Requirement Recommendation	Extraction out of documents	New	2.75 4 users voted

Figure 9.2.: Overview of an example project in OPENREQ!LIVE. Requirements (they consist of a unique ID, title, description, and status) are listed on the page and ordered by a utility value. Each release has constraints such as the deadline of the release and a maximum capacity value (in hours) which limits the possible number of requirements that can be assigned to the release. The OPENREQ services (see Section 9.3.3) have been integrated into the user interface. For example, the red labeled numbers indicate issues (such as requirement duplicates, ambiguities in a requirement’s description text, etc.) reported by some of the services.

Section 9.3.3 for more details). Strongly diverging arguments assigned to the same interest dimension indicate strong disagreement and appear as a conflict in the interface. The involved stakeholders have to discuss the issue and resolve the conflict by reevaluating the requirement (i.e., providing improved arguments based on the outcome of the discussion). This leads to more information exchange between stakeholders and has a positive impact on the quality of the prioritization process (Samer et al., 2020). After the elimination of all conflicts, the system computes a utility / priority value for each requirement using MAUT and ranks the requirements based on their priority (see Figure 9.2).

Besides OPENREQ!LIVE, a plugin for the ECLIPSE open-source community has also been developed within the scope of OPENREQ. The plugin aims to bring RE for open-source communities to the next level by helping the communities to reduce time and effort in the smart assignment of their requirements / issues to suitable developers as well as to attract many newcomers in onboarding scenarios. The plugin runs inside the ECLIPSE IDE⁷ (*integrated development environment*) which is the favored IDE of most developers in the community, and fetches requirements / issues relevant for the user / developer from OPENREQ’s *Issue Prioritizer Service* (see Section 9.3.3). The service connects to the web API of the issue tracking platform BUGZILLA⁸ and extracts new unresolved issues / requirements from the ECLIPSE project. The plugin is available for download on the ECLIPSE MARKETPLACE⁹

⁷ECLIPSE IDE: <https://www.eclipse.org/ide>

⁸BUGZILLA: <https://www.bugzilla.org>

⁹ECLIPSE plugin: <https://marketplace.eclipse.org/content/openreq-eclipse-ide-bug-prioritizer>

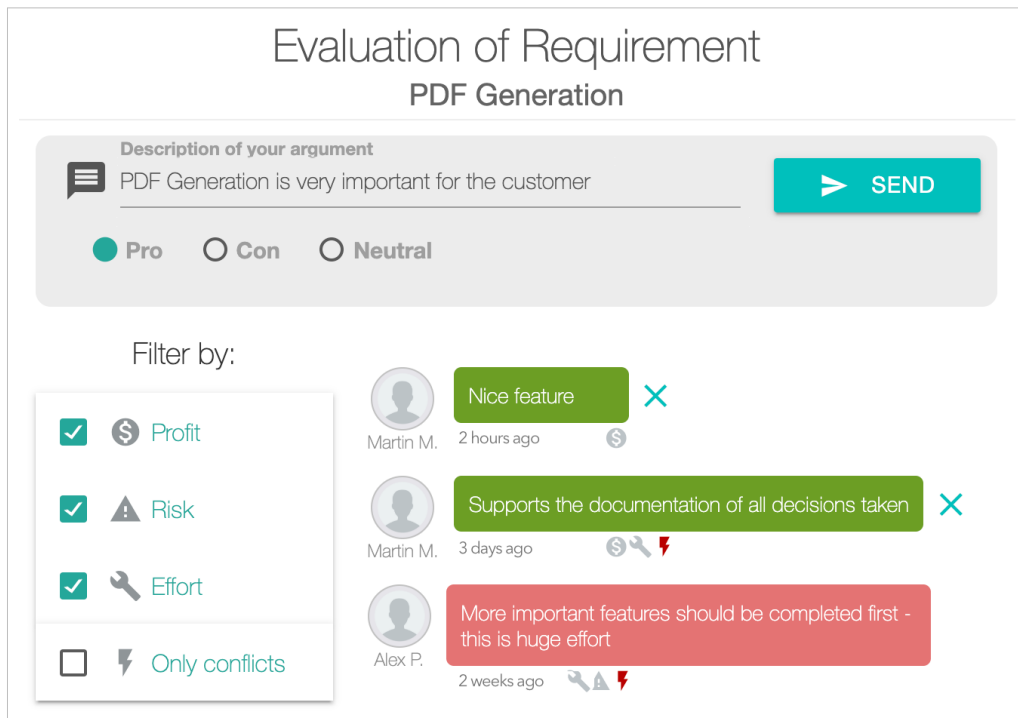


Figure 9.3.: Argumentation-based rating interface which allows stakeholders to exchange arguments for / against a requirement. Each argument must be assigned to one interest dimension. Negative arguments are highlighted in red, positive arguments in green, and neutral arguments in orange.

and provides a graphical user interface that shows the recommended requirements / issues to the developers (see Figure 9.4). Moreover, the plugin also allows developers to give feedback on each recommendation. Developers can give feedback in three different ways: *like button* (indicates that the recommendation was helpful), *dislike button* (indicates that the recommended issue is not relevant for the developer), and *snooze button* (indicates that the issue is not of highest relevance at the moment and the developer should be reminded about the issue again a few weeks later). Our recommendation service uses the individual feedback provided by the developers as additional input, in order to further improve the prediction quality.

To summarize, the developed user interfaces have large potential to enhance and speed up RE by making intelligent technologies developed in OPENREQ accessible in a user-friendly fashion for software development institutions as well as open-source communities. All user-interfaces are applicable on different platforms, such as Windows, Linux, and MacOS. In addition to that, OPENREQ!LIVE runs on a wide variety of web browsers and also supports various computing devices including different mobile and desktop devices.

ID	Summary	Priority	Product	Component	Dislike	Snooze	Like
516326	Add test case for bug 516304	100%	Platform	UI	👎	🕒	👍
387714	[CSS] Tag changes are not propagated as CSS class	66%	Platform	UI	👎	🕒	👍
458015	[CSS] Navigator not showing file	47%	Platform	UI	👎	🕒	👍
502581	Support local CSS changes by the end user	43%	Platform	UI	👎	🕒	👍
531138	Mark org.eclipse.ui.commands.ICommand, IHandler and related API for deletion	39%	Platform	UI	👎	🕒	👍
539592	Set CSS class to custom widget	39%	Platform	UI	👎	🕒	👍
474832	[Tests] Move o.e.ui.tests to JUnit 4	38%	Platform	UI	👎	🕒	👍
436183	[CSS] Revisit creating preference nodes in CSS bridge	37%	Platform	UI	👎	🕒	👍
516119	Cannot open the customized perspective dialog	35%	Platform	UI	👎	🕒	👍
521725	Several Platform UI tests are failing on Mac	35%	Platform	UI	👎	🕒	👍

Figure 9.4.: ECLIPSE plugin that recommends relevant requirements / issues to the active developer using content-based filtering based on supervised learning.

9.5. User Studies and Benefits

In this section, we present representative evaluations and study results of a narrowed list of selected empirical and user studies. All studies have been conducted within the scope of the OPENREQ research project and follow the purpose to evaluate the performance, usability, or prediction quality of the developed tools and approaches which have been discussed in Section 9.3. Most of the presented user studies were conducted in real-world scenarios with industry partners and some of our evaluated recommendation tools are also currently used by these industry partners.

Issue Prioritizer Service I (Section 9.3.4). We conducted a small-scale usability study (N=11 participants) with the developed ECLIPSE Plugin that calculates individual issue lists for the contributors of the ECLIPSE project by taking into account the individual preferences and the history. The persons who participated in our study were developers of the public ECLIPSE open-source project and 7 out of 11 participants (63.63%) either rated the usability to be good (2 participants) or even excellent (5 participants). Moreover, the quality of the results of the plugin satisfied all of the participants' expectations (5 participants (45.45%) stated that the recommendations were helpful and for 6 participants (54.55%) the recommendations were very helpful). By using the plugin, the participants of the user study could also take a look at their keyword profile. All participants stated that suitable keywords were found and the list of recommended requirements / issues was very accurate. Moreover, the participants were asked to estimate their perceived time savings in finding suitable requirements by using the plugin (compared to manual finding) within a range between 0 and 100, whereby 0 refers to no time savings and 100 refers to significant / high time savings. The results show that the perceived individual time savings were estimated very high on average (average: 74.18, median: 91.00, standard deviation: 31.92). Regarding future improvements, some of the participants mentioned that they would also see huge benefits if the developed tool could also be connected to other issue tracking systems (such as GITHUB issues or JIRA), in addition to BUGZILLA.

Issue Prioritizer Service II (Section 9.3.4). To demonstrate the potential of the content-based recommendation approach used by our *Issue Prioritizer Service*, we trained and tested our tool with

different classifiers (*Naïve Bayes* (our baseline), *Decision Tree*, and *Random Forest*). We used three large issue / requirement datasets of different open-source projects (ECLIPSE (N=141117), MOZILLA (N=751961), and LIBREOFFICE (N=47542)) to compare the performance of the different classifiers. The models used optimized hyperparameter combinations determined via *grid search*. In sharp contrast to already existing issue recommendation approaches, our approach is generally applicable which means that it does not focus on a single target group (such as newcomers). Considering this difficulty, the results show that our recommendation approach performs significantly better with *Random Forest* classification (in case of ECLIPSE, precision: 0.88, recall: 0.55, f1-score: 0.68) than already available and comparable approaches. Furthermore, the approach based on *Random Forest* classification also considerably outperforms our *Naïve Bayes* baseline (in case of ECLIPSE, precision: 0.53, recall: 0.29, f1-score: 0.38) as well as the simple baseline of the best random predictor for this specific recommendation task (in case of ECLIPSE, precision: 0.10, recall: 1.0, f1-score: 0.18). Note that we also compared our approach with a standard content-based approach that searches for k-nearest neighbor requirements based on different similarity metrics (e.g., cosine similarity). However, the results of this standard approach were quite poor and only slightly above the best random predictor baseline. For this reason, we did not include the standard content-based approach as baseline and decided to use *Naïve Bayes* which represents a more reliable baseline for this specific recommendation task. Further detailed results and the scores achieved on the other two datasets (MOZILLA and LIBREOFFICE) are presented in Samer et al. (2019). The bottom line is that the developed approach represents a good orientation towards future work which will focus on further improving the performance by using deep learning.

Argumentation-based Rating Interface (Section 9.3.3). To evaluate our argumentation-based rating approach, we conducted a large-scale user study with N=313 students in an RE-related course at our university (Samer et al., 2020). The students worked in groups of 4–6 students (60 teams) and each team developed a tourist information software. All teams had to use OPENREQ!LIVE to maintain their software project and apply OPENREQ!LIVE’s prioritization functionality (see also Section 9.4 and Figure 9.2). In this study, we compared three different prioritization approaches / versions: a *one-dimensional rating interface* (baseline), a *multidimensional rating interface* (MAUT-based version), and our *argumentation-based rating interface* (see Figure 9.3). We randomly assigned 20 teams to each rating version and disguised all requirement ratings / arguments of other team members until the current user evaluated the requirement to avoid psychological (cognitive) biases related to the hidden profile theory (Felfernig et al., 2018; Tversky and Kahneman, 1975). To counteract further biases related to anchoring effects, we did not inform the 60 groups and our 4 study assistants (who were responsible for the final assessment and grading of the student projects) about the existence of different UI variants during the study (Schulz-Hardt et al., 2006; Tversky and Kahneman, 1975) (double-blind). In order to ensure comparability of the study results and to compute a utility / priority value in each rating version, we used equal rating scales in all three versions. The one- and multidimensional rat-

ing versions allowed the students to rate a requirement in the range between 1 and 5 points. In the case of arguments, there existed three different sentiment levels (positive, neutral, negative) and we assigned 5 points to positive, 3 points to neutral, and 1 point to negative arguments. The evaluation results show that the argumentation-based rating interface helped to significantly increase the communication and interaction rate (includes created ratings and adaptations of existing ratings) among participants in the context of requirements prioritization. For example, groups that were confronted with the argumentation-based rating interface had 10.4 interactions per requirement on average (standard deviation: 1.03), whereas the interaction rates of the groups using the multidimensional rating interface (avg.: 5.4, std. dev.: 0.65) as well as the one-dimensional version (avg.: 4.5, std. dev.: 0.61) were significantly lower. The comparison of the grades the teams achieved at the end of the course and the quality of the developed software also reveals interesting differences / findings. The results of our analysis indicate that the teams that used the argumentation-based and multidimensional rating interfaces performed significantly better than our baseline (the one-dimensional version) in terms of grades and software quality (for more details, see Samer et al., 2020). Even though the explanatory power of the evaluation results is partly limited due to the university setting (the study was conducted with students in a university course), the results are very significant and expressive which helps to draw the conclusion that argumentation-based rating interfaces foster information exchange among stakeholders and have a positive impact on the quality of requirements evaluation and prioritization.

OPENREQ Classification Service (Section 9.3.1). The classification of whether a piece of text defines a requirement (REQ) or represents a textual description (PROSE) of a requirement is treated as a binary classification task. The utility of the *OPENREQ Classification Service* using this binary classification approach, was evaluated with ten datasets of different projects (30,000 requirements altogether). The datasets consist of samples (labeled as REQ and PROSE) which represent requirements and descriptive paragraphs of real-world industry projects. All datasets have been cleaned from irrelevant information, manually labeled, and evaluated by a small group of requirements managers and RE experts working at SIEMENS in Vienna (our industry partner). The finally obtained datasets represent a solid ground truth that reflects correctness and clearness of the data for the training of a reliable prediction model. Due to the nature of the datasets, there existed a high class-imbalance between REQ and PROSE classes, whereby REQ represented the majority class in all datasets (around 90%). The labeled datasets were used to evaluate our requirement classification tool. Our partner SIEMENS perceived the achieved level of prediction quality very promising (considering the highly imbalanced data) and thus integrated the developed approach into their (RE-related) business processes used in their real-world commercial operations (see Falkner et al., 2019) for detailed results). Hence, the primary outcome of this work is that the application of the *OPENREQ Classification Service* can lead to a significant work reduction for requirements managers and experts during requirements elicitation.

Orchestration Service (Section 9.3.1). The underlying approach was evaluated in a series of classification experiments. In these experiments, classifiers based on traditional supervised machine learning as well as deep learning were trained to find requirements-relevant information in English and Italian tweets as well as in English app reviews. All classifiers were trained to distinguish between the classes *problem reports*, *inquiries*, and *irrelevant* (see Section 9.3.1). The Twitter datasets consisted of labeled tweets (around 10,000 in English and 15,000 in Italian) from the telecommunication domain and the app review datasets consisted of around 6,000 app reviews. The learned prediction models performed well on all datasets. The best results were achieved on the dataset consisting of English app reviews (*problem report*: precision: 0.83, recall: 0.75, f1-score: 0.79; *inquiry*: precision: 0.68, recall: 0.76, f1-score: 0.72; *irrelevant*: precision: 0.88, recall: 0.89, f1-score: 0.89). In general, the results indicate that, within the used experimental settings, the approaches based on traditional machine learning could achieve comparable results to deep learning. For more detailed results, we refer to Stanik et al. (2019).

9.6. Related and Future Work

Related Work. In order to address the problems discussed in Section 9.2, a lot of research has been conducted with respect to the aforementioned aspects. Different scientific works identify a need for intelligent tool support to help requirements engineers and stakeholders in the different stages of the RE process for complex projects (Castro-Herrera et al., 2009; Mobasher and Cleland-Huang, 2011; Ninaus et al., 2014). A recommendation approach, for example, may be helpful to suggest requirements to a stakeholder, who already dealt with the same topics in the past. Alenezi et al. (2013) present a content-based approach which is used to predict and recommend relevant bugs based on the experience of the stakeholder. As for finding dependencies between requirements, existing research shows that *natural language processing* (NLP) techniques can be applied (Chitchyan and Rashid, 2006; Deshpande, 2019). Although there already exist several tools for parts of the RE process, the unique nature of the projects makes it rather complex to find methods which are valid for every project. Ninaus et al. (2014) present a small RE platform called INTELLIREQ which applies recommendation techniques to support stakeholders in different RE tasks. INTELLIREQ goes one step further and aims to tackle this challenge by providing intelligent tools, approaches, and techniques for different RE scenarios to the RE community. INTELLIREQ utilizes basic recommendation techniques to support stakeholders in common RE tasks. However, when compared to OPENREQ's RE platform OPENREQ!LIVE, INTELLIREQ only provides a small and limited set of very basic features which do not go beyond the level of semi-automated learning.

Future Work. The OPENREQ!LIVE user interface will be continuously improved in terms of integrating functionalities to automatically annotate, evaluate, and group requirements as well as to open its application for existing requirements engineering tools (such as IBM DOORS) by providing inter-

faces to such systems. Furthermore, text processing techniques are still an active research topic which receives contributions from several domains. In our future work, we plan to focus on this issue and compare some of our developed content-based recommendation approaches with more sophisticated approaches based on deep learning.

9.7. Conclusion

This chapter provided an overview of new RE recommendation tools developed in the context of the European research project OPENREQ. The RE process can be viewed as a comprehensive decision-driven process consisting of different phases in which many stakeholders are involved. A high complexity in this process implicates a high risk of project failure. In this context, we presented the OPENREQ project as an example of a research project in the field of RE. We introduced the OPENREQ approach which addresses the aforementioned issue and presented OPENREQ's RE recommendation tool suite consisting of innovative recommendation solutions which are applicable in different RE-related tasks. The variety of intelligent technologies and services shown in the chapter fit seamlessly into OPENREQ!LIVE, which is an RE platform that provides a central interface for stakeholders to access the most relevant OPENREQ services in a user-friendly way. The OPENREQ project is important, not only for tool support, but its user studies can guide research in directions that are supported by actual real-world cases. The OPENREQ approaches are supported by the evaluation of OPENREQ methods by industry partners from three different areas (*railway safety systems, telecommunications, distributed open-source development*). The evaluation results clearly prove the point that the developed concepts are able to significantly improve the RE process, in terms of more information exchange among stakeholders, lower costs, reduced time effort, and increased quality of the process output.

Conclusions & Future Work

In the field of software development, *requirements engineering* is a very decision-driven discipline where decision quality has a major impact on the success of a software project. This triggers the need to apply intelligent systems in requirements engineering to support stakeholders in critical decision tasks. *Recommender systems* are proven technologies to support various aspects of decision-making tasks (Quadrana et al., 2018), for example, prioritization tasks or the identification of preferences. With the help of recommender systems, stakeholders can focus on more productive tasks and thus make better decisions. Recommender systems have the potential to identify the most relevant information pieces within complex requirements models in order to provide enhanced decision support for stakeholders in critical decision tasks. This has motivated and inspired us to put our main research focus on this topic. Within the scope of the European research project OPENREQ¹ (part of the European Union Horizon 2020 program), we have developed a requirements engineering platform called OPENREQ!LIVE. The OPENREQ!LIVE platform provides extensive decision support for the most relevant requirements engineering tasks. OPENREQ!LIVE includes and features several recommendation approaches that address various tasks of the requirements engineering process. In this chapter, we reflect on the defined research questions (see Section 1.2) and we highlight the major contributions that are related to our research questions. Finally, we discuss the limitations of our recommendation approaches and provide an outlook for future research to conclude this chapter.

10.1. Conclusions

This section presents a summary of answers to the research questions which were defined in Section 1.2. The discussion of these answers summarizes our approaches as well as our major challenges that we have faced in our experiments and evaluations. Furthermore, we outline the major outcomes and findings of our research work.

¹OPENREQ: <https://openreq.eu>

Research Question Q1.1:

How can we automatically identify dependency relations between textually-defined requirements?

During a first in-depth analysis of task areas for which only a few mature decision-support approaches exist, we were able to identify the area of *requirement dependency detection* as an initial starting point for our research work. The complete awareness of all dependency relationships which exist between the requirements in a software project is crucial for release planning, and provides the ability to detect potential redundancies and inconsistencies in a requirements model (Aguilar et al., 2012). In Section 4.5, we present two content-based recommendation systems which advance the state-of-the-art by achieving a high prediction quality in the automated identification of requirement dependencies. Both systems focus on the dependency type *requires* since this type is the most critical type among all existing dependency types (Ferber et al., 2002). The first system was based on supervised classification algorithms (*Naïve Bayes*, *Linear Support Vector Machines*, *k-Nearest Neighbors*, and *Random Forest*) and the second was based on *Latent Semantic Analysis* (it defined our baseline for evaluation). Both systems were evaluated with a real-world dataset consisting of 30 requirements. The dataset was collected in an industry project. In order to assure high data quality for training and to obtain a reliable ground truth, we have extended the dataset with dependencies in the scope of a comprehensive user study which we have conducted together with industry experts (see Section 4.3). The full dataset was then used to train and evaluate our two recommendation systems – both with *TF-IDF features* and the first one also with *probabilistic features*. While TF-IDF features only take into account how often the individual words occur in the description text of the requirements, the idea of probabilistic features follows the approach of sensing word-related correlations between different requirements. The main challenges that we faced at the beginning were related to the small size of the requirements' text and the risk of overfitting our recommendation models. By applying special mechanisms (such as hyperparameter tuning and under-sampling of the training samples), we were able to overcome these hurdles. Section 4.6 presents our evaluation results, which provided evidence that our first recommendation approach performed reliably well with different classifiers on the given dataset. Moreover, the recommender that used a Random Forest classifier trained with probabilistic features, was able to outperform our baseline significantly. Our main finding is that probabilistic features represent an effective means to improve the prediction quality in this evaluation setting (with short requirement description texts).

Research Question Q1.2:

What is a proper solution to handle cold-start issues in requirements dependency identification tasks?

When using recommender systems to support the identification of requirement dependencies, the *cold-start problem* often represents a serious challenge whenever there are not enough data records of other domain-related projects available. As part of this work, we present first steps on how to tackle the cold-start problem in the context of requirement dependency identification. To that end, we have implemented a recommendation approach (see Section 4.5) that exploits *Latent Semantic Analysis*² to detect requirement dependencies based on the similarity of semantic characteristics extracted from the text of requirements. To simulate the cold-start problem, we trained our approach with a real-world dataset that only contained the textual content of the requirements. We compared our approach with several classification-based algorithms and used the dependency-related data to measure the prediction quality. During the evaluation we have observed that our approach sometimes mistakenly considered very similar requirements to be interdependent. Due to this reason, we added a hyperparameter to our approach, which prevents the approach from considering very similar requirements as being dependent. As mentioned in Section 4.6, the evaluation of our recommendation approach reveals that our approach is able to discover a fairly large number of correct dependencies which is reflected by a high recall score and a good overall prediction quality. The main outcome of our evaluation is that our approach represents a good orientation towards more sophisticated solutions to overcome cold-start problems in the context of requirement dependency identification tasks.

Research Question Q2.1:

How does the dimensionality of rating-schemes affect requirements evaluation behavior?

Beyond providing recommendation support to automatically identify requirement dependency relationships, we could determine the *prioritization of requirements* as another important task for which there still exists much potential for improved decision-support. The prioritization of requirements is usually based on group-based evaluation processes. Many existing prioritization techniques do not direct the evaluation process towards a multi-aspect oriented evaluation approach, but rather convey the impression that requirements should be viewed as single abstract evaluation units. This can lead to suboptimal situations where stakeholders tend to evaluate requirements on the basis of one dimension and are not encouraged to assess them individually based on different evaluation criteria. This problem defines our next interesting research subject of this thesis. To that end, we implemented three user interfaces equipped with different rating-schemes (a basic one-dimensional, a basic multi-dimensional, and an argumentation-based, multidimensional rating-scheme) to evaluate and prioritize requirements. A group recommendation approach was used to recommend a prioritized list of requirements based on the stakeholder evaluations. The group recommendation approach and the user interfaces are described in Section 5.3. In order to examine the stakeholder evaluation behavior when using one of these rating interfaces, we conducted a large-scale user study with 313 computer sci-

²Note that this approach also served as a baseline to answer research question Q1.1.

ence students who worked in small groups of 4–6 students to develop a software application. Each group was exposed to one rating interface (between-subject design (Charness et al., 2012)) and had to use the rating interface to define, evaluate, and prioritize their requirements. In our empirical study, a 5-star rating interface was used as a basic one-dimensional rating-scheme, and both multidimensional rating-schemes included three interest dimensions (financial profit, project risk, and technical effort) to introduce a higher level of depth to the evaluation process. The evaluation results (see Section 5.4) reveal that basic one-dimensional rating schemes can affect the stakeholder behavior such that the stakeholders tend to be very optimistic about the feasibility and the implementation of the requirements. In contrast, groups who use the multidimensional rating interfaces showed more realistic estimations, which is reflected by a lower number of requirements and more critical requirement ratings (observed as lower rating values). The main outcome that can be inferred from these results is that multidimensional rating interfaces can make a significant contribution to better prioritization results, which the following research questions attempt to confirm.

Research Question Q2.2:

How can we increase stakeholder interaction in requirements evaluation to improve the quality of requirements prioritization?

Group decisions are often dominated by a small group of people. In the context of requirements prioritization, this can negatively affect the evaluation and prioritization of requirements, especially when opinion leaders express their opinion first and thus steer the opinions of the rest of the people in a certain direction. This psychological effect represents a decision bias that is known as the *anchoring effect* (Mojzisch and Schulz-Hardt, 2010; Stettinger et al., 2015; Tversky and Kahneman, 1975). In addition, the early disclosure of views by decision-makers can make other stakeholders remain more passive. To counteract and mitigate these effects, we have implemented three group-based recommendation approaches which are presented in Section 5.3. All three approaches hide the other stakeholders' ratings until the stakeholder has rated the requirement. One approach uses a basic one-dimensional 5-star rating interface and the other two approaches use a multidimensional rating interface. One multidimensional approach follows the objective of triggering more information exchange among stakeholders by empowering them to provide ratings in the form of textual comments. Every comment represents an aspect-related argument for (positive) or against (negative) a requirement. Within the scope of an extensive user study, we evaluated and compared all three recommendation approaches. Our related research results are presented in Section 5.4. The results show that by introducing argumentation-based assessments, we could encourage the participants to evaluate and prioritize their requirements more intensively. While the one-dimensional ratings were rarely adapted by the participants and therefore often led to a fast decision convergence within the groups, a more dynamic decision-making behavior could be observed for groups who used the multidimen-

sional recommendation approaches. The most positive results, however, could be achieved with the argumentation-based approach, where the arguments of the participants led to more subsequent rating adjustments / adaptations and to a more opinion-driven decision-making process. More rating adaptations are an indication for increased information exchange, which is a major precondition for high-quality group decisions (Schulz-Hardt et al., 2006; Greitemeyer and Schulz-Hardt, 2003).

Research Question Q2.3:

In which way do different evaluation interfaces impact requirements prioritization and software quality?

Various research works (Boehm, 1981; Davis, 2005; Cleland-Huang et al., 2003) show that faulty, careless, or delayed requirements engineering represents a serious problem that has a negative effect on the software quality and the development costs of the software project. Boehm (1981) states that around 60 percent of development errors originate from this problem (see also Section 1.1). In order to achieve a better software quality, one possible first step is to improve the prioritization of requirements to identify and focus on the most relevant requirements early in the project. Requirements prioritization helps to improve quality in several places. For example, this allows stakeholders to identify important features. Furthermore, the maintainability of the software can be increased and thus the susceptibility to errors can be reduced since it is no longer necessary to add so many requirements during the development of the software. An example would be the integration of a "redo / undo" functionality into an *already* existing software product. In this thesis, we present a group recommendation approach that includes three different user interfaces to evaluate and prioritize requirements (see Section 5.3). The user interfaces used a basic one-dimensional, a basic multidimensional, or an argumentation-based multidimensional rating scheme. To examine the impact of the user interfaces on the software quality, we conducted a large-scale user study with computer science students who had to work in groups to define and prioritize requirements in order to develop a software project based on the prioritized list of finally selected requirements. To assess the software quality, we measured the number of successfully completed requirements and the grades (grading points) the students have received for the software product at the end of the study. As mentioned in Section 5.4, the results of our empirical study indicate that requirements prioritization approaches supported by user-interfaces with a multidimensional rating scheme can increase the software quality. In addition, slight but no significant differences of software quality could be observed between the argumentation-based and basic multidimensional rating schemes. However, a better prioritization outcome achieved with our argumentation-based approach (indicated by more rating adaptations and more information exchange through arguments; see Q2.2) tends to lead to reduced costs and improved implementation schedules due to the neglect or elimination of unnecessary requirements (Achimugu et al., 2014; Firesmith, 2004).

Research Question Q3.1:

How can we facilitate stakeholder assignment to support decision-makers?

The prioritization of requirements is often regarded as a preliminary task which is followed by the assignment of requirements to stakeholders. In the context of traditional requirements engineering, the identification of suitable requirements to assign to stakeholders represents a complex decision task. Inappropriate assignments can lead to situations where the stakeholders working on the requirements are not those who are the most suitable candidates to take over responsibility for the requirements. This can lead to serious consequences caused by programming errors (see Section 1.1). However, the late detection of software errors poses a threat to the project (e.g., missed deadlines, increased costs, project failure). Today's software projects typically consist of many requirements. Nevertheless, in traditional requirements engineering scenarios only a few people (requirements managers) are responsible for the identification and assignment of requirements. In Section 6.5, we present a novel recommendation approach to distribute the stakeholder assignment task among several stakeholders. Our recommendation approach consists of two different recommendation services that support the stakeholders during the entire decision-making process of the stakeholder assignment task. A content-based recommendation service proposes potential stakeholder candidates for the requirements. Stakeholders (including requirements managers) can evaluate / rate these candidates and propose further candidates. A group-based recommender ranks the candidates based on the evaluations and suggests a final candidate to the requirements managers. Our approach represents a first step towards a more flexible working environment that improves this complex stakeholder assignment task by reducing the overall involvement of requirements managers. The automated recommendation of suitable requirements to stakeholders has the potential to reduce software weaknesses (such as bugs, other software defects, or immature features) in the final software end product, counteract time delays, extra costs, or even avoid project failure (in the worst case) (Lim et al., 2010; Bhattacharya et al., 2012; Pacheco and Garcia, 2008; Pacheco and Tovar, 2007). Moreover, our approach facilitates the work of requirements managers by distributing the stakeholder assignment task to different stakeholders in order to reduce the time effort of the requirements managers as well as the risk that important stakeholder assignments are overlooked.

Research Question Q3.2:

How can we foster stakeholder engagement in open-source development while taking into account different stakeholder types (e.g., newcomers and experts)?

To create a recommendation environment that supports stakeholders in the development of open-source software, it is crucial to provide assistive guidance for stakeholders in order to relieve them of certain standard "minor" tasks such as "finding the next requirement to work on". Most exist-

ing tools to support stakeholders in the development phase of a software project were created for working environments where developers work in hierarchies (such as in companies or organizations) and are not expedient for use in open-source communities where often no clear (or only rudimentary) stakeholder hierarchies exist. In Section 8.3, we introduce a content-based recommendation approach that represents a first step to bridge this gap. Our approach addresses large open-source communities and is based on machine learning (supervised classification). It supports open-source contributors in the finding and prioritization of relevant requirements (in large requirement databases) by suggesting a ranked list of requirements to work on next. We have developed a plugin for the ECLIPSE community that presents a ranked list of recommended requirements to the stakeholders and allows them to interact with the recommendations. To measure the prediction quality of our approach, we evaluated three classifiers (Multinomial Naïve Bayes, Decision Tree, Random Forest) on three large datasets from open-source projects (ECLIPSE, MOZILLA, LIBREOFFICE). An important step to increase stakeholder engagement for different stakeholder types was to focus on the most relevant recommendations (*quality over quantity*). By following this objective, we could develop a strategy to tune the hyperparameters of our recommendation model in order to obtain higher precision rates rather than recall scores (quality over quantity). In combination with the Random Forest classifier, we could gain a model achieving a high prediction quality that significantly outperforms all baselines (see Section 8.4). The major challenges that emerged in this context were, on the one hand, the variability of the developer’s preferences, which can change over time, and, on the other hand, the difficulty of assigning many developers to the different work packages. We solved this by dividing the developers into subject areas (“components”) and by creating a separate user profile for each component in which they were active. The user interface of our ECLIPSE plugin enables the developers to select the subject areas (or a combination of these) for which they want to receive recommendations. To evaluate the usability of our user interface as well as the perceived quality of the recommendations, we conducted a small-scale usability study with 11 developers from the ECLIPSE community. The results of our study are presented in Section 9.5. The study results reveal that all participants were satisfied with the recommendation quality and the majority of them (more than 63%) rated the usability of the plugin to be good or very good. Moreover, the time savings in finding suitable requirements were perceived to be very high by most participants when compared to manual finding. Given these results, we can conclude that our approach has the potential to facilitate the stakeholder assignment process in open-source communities. Furthermore, our recommendation approach optimizes the overall productivity of open-source developers.

Limitations

The recommendation approaches and studies presented in this thesis have some limitations that provide a good orientation towards future work and are briefly discussed at this point of the thesis. One limitation relates to the evaluation of our developed recommendation approaches where our approaches were evaluated within the scope of only one specific domain. A reevaluation of our

approaches for different domains may lead to small deviations of our evaluation results. The presented recommendation solution for detecting requirement dependencies has only been evaluated with a small dataset from one domain. Although the evaluation results give a first impression of the applicability of our classification-based recommendation approach using probabilistic features, the developed approaches were not designed for cross-domain application scenarios. Another limitation relates to the recommendation service, which recommends suitable requirements to open-source software developers. Due to the large size and complexity of many open-source software projects, the projects are divided into different subject areas (components). Our recommender follows the approach to create different component-specific user profiles for the developers. On the one hand, this leads to the limitation that the recommendation algorithms cannot suggest requirements from other components in which the developers were not active in the past (i.e., have not resolved any requirements of the component). On the other hand, this also gives the developers the option to specify the project components for which they would like to receive recommendations. Regarding our group recommender user interfaces to support requirements prioritization scenarios, we can identify a few more mentionable limitations which are related to our user study. One limitation was that all study participants were computer science students and the majority of participants were male students (around 10% female and 90% male participants). Moreover, in each student group, one student was responsible for the administration of the software project (stakeholder role: project manager) and all other students participated in the project as a software developer (stakeholder role: software developer). Limitations of the selected study setting relate to small group sizes (4–6 students) and to the domination of one stakeholder role (i.e., software developer). Apart from a few other participants who were project managers, no other stakeholder roles were considered in our study. Consequently, the participants can be viewed as a relatively homogeneous group in an evaluation scenario, which does not ideally reflect the conditions of a real working environment in a large company or organization. Further limitations are the investigation of mechanisms for resolving rating conflicts (strongly differing requirement evaluations of different stakeholders), as well as a textual analysis of the user-generated arguments and requirements (e.g., measure the impact of the text length and textual ambiguities on the final prioritization result).

10.2. Future Work

Based on the outcome of our research work and the limitations discussed (see Section 10.1), we can identify some interesting topics regarding future work. The remainder of this section presents a discussion of further relevant future research topics.

Further Investigations and User Studies

The evaluation results of our research works represent promising contributions which have the potential to guide and advance requirements engineering research in the future. The continuous extension

of the presented recommendation approaches represents one key part of future work. In this context, the focus must also be shifted towards a detailed investigation of further application domains. In addition, more extensive usability studies with higher numbers of participants and requirements would be useful to improve the adaptation of the existing recommendation technologies to the way people work and how they use these decision-support technologies. Future studies should also consider a more balanced ratio of male and female participants, and include different types of stakeholders from the industry (for example, project managers, customers, product designers, and software developers). Moreover, larger datasets consisting of labeled requirement dependencies are needed to overcome some limitations mentioned in Section 10.1. These datasets can be collected using the crowd knowledge of industry experts working in the field of requirements engineering. Existing requirement dependency datasets are often incomplete or contain unverified assumptions of dependency relationships that do not represent a solid ground truth for recommendation solutions based on machine learning predictors. In order to collect new large datasets, we propose to conduct further user studies that involve many industry experts (as well as different types of stakeholders) from various branches of industry. Beyond that, the use of *weakly supervised learning* (see initial work presented by Deshpande et al., 2019) represents an innovative approach to extend existing requirement data with more artificially generated dependency labels. Weakly supervised learning helps to significantly reduce the querying of human experts to label a large list of requirements, which is very costly. In the future, we plan to follow a more hybrid approach that uses different classifiers, stakeholders, and experts to identify dependencies. Experts should analyze significantly fewer requirement pairs and primarily focus on those dependencies for which pre-trained classifiers and different stakeholders have provided strongly diverging estimations.

Regarding our user study to compare the group recommendation user interfaces, the investigation of the textual content of the requirements and arguments represents another interesting idea for future work. For example, we could analyze the ranking of similar requirements in order to estimate the utility of these requirements based on a statistical comparison (e.g., measure the correlation of the words in the description of different requirements). This has the potential to improve our group recommendation solution such that the recommender can detect newly defined requirements and propose a rating which is based on former ratings of similar requirements from past projects. Moreover, a small text length and ambiguities in the text of arguments or requirements are often a strong indication for poorly formulated requirements / arguments (Bano, 2015; Kamsties, 2005). This can deteriorate the understandability of the requirements which can negatively affect the requirement ratings and consequently the prioritization results as well. The aforementioned aspects of detecting poorly phrased requirements / arguments can be taken into account by future recommendation solutions in order to improve the quality of the final prioritization.

Advanced Technologies

The fact that most of the tasks in requirements engineering can only be done by humans complicates the usage of intelligent techniques. Consequently, the definition and elicitation of requirements, negotiation of requirements, and implementation of requirements, as well as quality assurance, are still responsibilities for humans, no matter which technique was chosen. As part of this thesis, we presented some innovative recommendation approaches to tackle these issues and provide decision-support to stakeholders in some of these tasks. The final outcome of the thesis represents a first step towards advanced decision-support in requirements engineering. Based on this work, we propose the use of more advanced technologies to further improve the prediction quality and to eliminate further sources of error. Since requirements are very often described in natural language and much progress has been made in the field of *natural language processing* with *deep learning* over the last several years (Deng and Liu, 2018; Otter et al., 2020), the use of recommendation technologies based on *deep neural networks* represents a proper subject for future investigations. In particular, the integration of deep neural networks into the existing recommendation solutions can help to further improve the overall recommendation quality of the presented recommendation approaches.

Moreover, an increased use of *liquid democracy* can facilitate group decisions in various group decision tasks (Johann and Maalej, 2015; Zhang and Zhou, 2017; Atas et al., 2018). The underlying principle of liquid democracy is to allow stakeholders to transfer their votes (or parts of their votes) to experts whenever they do not feel confident to make the decision. This way, the transfer of voting power is controlled by each stakeholder individually and the transfer of votes to individual experts or expert groups happens in a democratic fashion. Liquid democracy has already been used in our evaluation of different group recommendation user interfaces. In this context, we plan further investigations on how to recommend suitable experts to transfer the vote for a specific decision task. Furthermore, we want to analyze the areas in which expert knowledge is distributed in the stakeholder community, in order to show project managers how the knowledge is distributed in a company and where critical parts of the business knowledge are located. This can improve decision-making in group decision tasks and allows the identification of "hidden champions" (talents / experts) in a company. Based on this knowledge it is possible to support the "hidden champions" in a personalized fashion.

Enhanced Preference Elicitation

One way to improve the quality of recommendations is to continuously adapt a recommender system to new user feedback regarding the perceived recommendation quality (Zhao et al., 2018; Zheng et al., 2018). For example, the perceived recommendation quality can be determined by allowing users to give direct feedback on individual recommendations. Our recommendation service which recommends requirements to ECLIPSE developers represents an excellent area of application in which a stronger integration of user feedback can help to better support the open-source developer community. The most recent version of our ECLIPSE plugin already allows developers to provide feedback

on recommended requirements in terms of *liking*, *disliking*, or *snoozing* (“remind me later about this requirement”) the recommended requirements (see Figure 8.1 in Chapter 8). This feedback represents important knowledge to further improve the recommendation service. At the moment, we use the feedback to re-rank the recommendation list or to hide certain requirements depending on the type of feedback (*like*, *dislike*, *snooze*). However, this work can be further extended in the future to continuously adapt the user profiles during operation time and to further improve the personality of recommendations as well as the overall recommendation quality.

Another idea for future research is the development of *configuration-based techniques* to facilitate the onboarding of new developers in open-source communities. This helps to strengthen the affiliation and commitment of new developers to keep contributing to the project. For example, our ECLIPSE plugin can be used to determine the preferences of newcomers more specifically. This can be achieved by means of the integration of configuration-based recommendation approaches and more specific preference and knowledge elicitation approaches that let newcomers answer a few initial multiple-choice questions (e.g., *What are your personal strengths and weaknesses?*, *What kind of tasks would you like to solve at the beginning?*, or *What tasks should be reassigned to other experts?*) which are intended to prevent unsuitable requirements from being recommended to the developer. Improved measures to avoid the recommendation of unsuitable requirements can lead to increased motivation of newcomers. Furthermore, this also helps to overcome *cold-start issues* by giving developers the opportunity to help build a more precise user profile.

Weighting of Opinions

The weighting of user opinions based on expertise represents a promising method to further enhance decision-making. In particular, this applies to requirements engineering tasks such as the *requirements prioritization* or *stakeholder assignment*. For example, when making group decisions to prioritize requirements, it often makes sense not to assign the same voting weight to every stakeholder, but to assign higher weights to experts or key decision-makers. Within the scope of future work, we want to extend our group recommendation user interfaces to learn stakeholder weights (based on interest dimensions such as profit, effort, risk) from historic data of past projects. For instance, experts can be identified from vote delegation trends inferred from past vote delegations (*liquid democracy*). Regarding stakeholder assignment in traditional software development, group decision processes can benefit from the automated assignment of weights to stakeholders for different interest dimensions depending on the department or the stakeholder role. By taking into account the participants’ feedback of our usability study (see Section 9.5), another research focus for future work is the manual customization of user profiles by using our ECLIPSE plugin. The basic idea is that experienced ECLIPSE developers can modify existing keywords and keyword weights of their user profile to further customize the recommendations according to the developers’ preferences.

Explainable Recommendations and Decisions

Good explanations of recommendations (in textual or visual form) represent an efficient method to increase satisfaction and trust of users in recommender systems (Tintarev and Masthoff, 2007). First preparations regarding explanations have already been made within the scope of this thesis. For example, our ECLIPSE plugin presents the most relevant keywords of a developer's user profile in a visual form to give the developer a rough idea on the basis of which the recommendations were generated. The existing work can be further extended in the future in several ways. Recommended requirements can be explained with relevant keywords extracted from the requirement which also appear in the user profile of the developer. Moreover, we want to extend our group-based requirements prioritization approaches and include explanations which present and highlight rating conflicts between different stakeholders. These explanations are supposed to trigger more information exchange among stakeholders in order to get more explainable and better results of group decisions.

Further Study Results of Chapter 5

This section includes additional statistical details of study results discussed in Section 5.4. Table A.1 provides key figures (in terms of the arithmetic mean / average and the standard deviation) of the measured data, including the number of evaluations (ratings) per requirement, the number of rating adaptations (interactions) per requirement, as well as the number of points achieved by the students. The data has been separated by the used group recommendation *user interface* (UI); *5-star rating* (Section 5.3.1) vs. *basic MAUT* (Section 5.3.2) vs. *argumentation-based MAUT* (Section 5.3.3).

UI type	avg. / stdev. interactions per requirement	avg. / stdev. evaluations per requ.	avg. / stdev. points per student
5-star rating	4.5 / 0.61	3.9 / 0.84	16.44 / 13.33
basic MAUT	5.4 / 0.65	4.2 / 0.92	22.20 / 10.78
arg.-based MAUT	10.4 / 1.03	5.1 / 0.99	24.31 / 8.64

Table A.1.: Study results of our group recommendation UI types (*5-star rating*, *basic MAUT*, *argumentation-based MAUT*) showing statistical details about the average (avg.) and standard deviation (stdev.) of the evaluation interactions (rating adaptations), the evaluations / ratings, and the points achieved by the students.

In order to further strengthen and confirm the expressive power of our findings, we have also evaluated the significance of the observed differences between the different user interfaces using the *Mann–Whitney U test* (Mann and Whitney, 1947) in addition to the *t-test*. In general, the *t-test* expects the analyzed data to be normally distributed (which was the case for some of our evaluated comparisons) but can also be considered as robust for large numbers of samples ($N > 30$) which are not normally distributed (which was the case for all of our comparisons; $N \gg 30$). The *Mann–Whitney U test* (also known as *Wilcoxon–Mann–Whitney test*) is a non-parameterized test which can be used as an alternative to the *t-test* if the data is not normally distributed (Hart, 2001). The Tables A.2, A.3, and A.4 present the pairwise comparisons of our group recommendation UI types with respect to statistical differences of observed values in terms of the number of ratings, the number of rating interactions,

and the number of points achieved by the students. All these comparisons have been evaluated using the t-test (see Section 5.4 as well as the Mann–Whitney U test (see Tables A.2, A.3, A.4). As can be seen in Section 5.4 and in the aforementioned tables, both test methods lead to similar results¹.

Samples X	Samples Y	U	p	Z	r	Stat. Significance
5-star rating	basic MAUT	101290.5	< 0.01	-4.59	-0.15	yes
5-star rating	arg.-based MAUT	32453.0	< 0.01	-15.55	-0.53	yes
basic MAUT	arg.-based MAUT	39010.0	< 0.01	-11.63	-0.41	yes

Table A.2.: Pairwise comparison of our group recommendation UI types (*5-star rating*, *basic MAUT*, *argumentation-based MAUT*) with respect to statistical differences in the number of ratings / evaluations using a non-parameterized statistical (stat.) significance test (*Wilcoxon-Mann-Whitney-Test*). The pairwise significance tests were calculated to determine if the diverging observations between the different UI versions are statistically significant (i.e., whether significantly more ratings / evaluations were observed in Y than in X).

Samples X	Samples Y	U	p	Z	r	Stat. Significance
5-star rating	basic MAUT	43644.5	< 0.01	-17.43	-0.55	yes
5-star rating	arg.-based MAUT	0.0	< 0.01	-24.70	-0.84	yes
basic MAUT	arg.-based MAUT	10.5	< 0.01	-23.98	-0.85	yes

Table A.3.: Pairwise comparison of our group recommendation UI types (*5-star rating*, *basic MAUT*, *argumentation-based MAUT*) with respect to statistical differences in the number of rating adaptations / interactions using a non-parameterized statistical (stat.) significance test (*Wilcoxon-Mann-Whitney-Test*). The pairwise significance tests were calculated to determine if the diverging observations between the different UI versions are statistically significant (i.e., whether significantly more rating adaptations / interactions were observed in Y than in X).

Samples X	Samples Y	U	p	Z	r	Stat. Significance
5-star rating	basic MAUT	3900.5	< 0.01	-3.06	-0.21	yes
5-star rating	arg.-based MAUT	4066.5	< 0.01	-3.66	-0.25	yes
basic MAUT	arg.-based MAUT	5112.0	0.42	-0.18	-0.01	no

Table A.4.: Pairwise comparison of our group recommendation UI types (*5-star rating*, *basic MAUT*, *argumentation-based MAUT*) with respect to statistical differences in the number of points achieved by the students using a non-parameterized statistical (stat.) significance test (*Wilcoxon-Mann-Whitney-Test*). The pairwise significance tests were calculated to determine if the diverging observations between the different UI versions are statistically significant (i.e., whether a significantly higher number of points was observed in Y when compared to X).

¹As a side note, it is important to point out that we have removed the outliers from the data samples before the calculation. In order to ensure that the removal of these outliers does not violate the significance of our results, we have also calculated the tests taking into account all outliers; this resulted in small deviations of the results which can be neglected as these deviations did not have any observable influence on the statistical significance of our results.

Detailed Evaluation Results of Chapter 8

This section presents detailed evaluation results of Section 8.4. The results are divided into several tables according to the evaluation metrics *precision*, *recall*, and *f1-score*. Since we optimized the hyperparameters of our recommendation models to focus more on precision (rather than on recall), the highest precision values in Table B.1 are highlighted in bold.

<i>Algorithm</i>	ECLIPSE	MOZILLA	LIBREOFFICE
Constant Predictor (Simple Baseline)	0.10	0.07	0.09
Multinomial Naïve Bayes (Baseline)	0.53	0.72	0.77
Decision Tree	0.62	0.45	0.67
Random Forest	0.88	0.73	0.81

Table B.1.: Precision scores of the different classifiers and the constant predictor baseline. The highest precision rates for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE).

<i>Algorithm</i>	ECLIPSE	MOZILLA	LIBREOFFICE
Constant Predictor (Simple Baseline)	1.0	1.0	1.0
Multinomial Naïve Bayes (Baseline)	0.29	0.15	0.24
Decision Tree	0.49	0.49	0.71
Random Forest	0.55	0.49	0.75

Table B.2.: Recall rates of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE).

<i>Algorithm</i>	ECLIPSE	MOZILLA	LIBREOFFICE
Constant Predictor (Simple Baseline)	0.18	0.13	0.17
Multinomial Naïve Bayes (Baseline)	0.38	0.25	0.37
Decision Tree	0.55	0.47	0.69
Random Forest	0.68	0.59	0.78

Table B.3.: F1-scores of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE) were achieved by Random Forest and are highlighted in bold.

List of Figures

2.1. Choice scenarios categorized with regard to (1) <i>constraint inclusion</i> and (2) the <i>representation of alternatives</i> (as parameters or items).	20
3.1. Liquid democracy. Example of a vote-delegation tree for a single attribute or a single requirement, respectively. All vote-transfers are visualized as (directed) edges and all stakeholders are represented as nodes.	56
3.2. Example of a group-based requirements evaluation scenario in INTELLIREQ (see Ninaus et al., 2014). The traffic light feedback mechanism indicates an inconsistency with respect to the three stakeholder ratings provided for the property <i>priority</i> (red light). .	73
3.3. OPENREQ!LIVE project overview.	74
3.4. Argumentation-based rating interface. Each argument must be assigned to at least one interest dimension. Negative arguments are highlighted in red, positive ones in green, and neutral arguments in orange.	76
3.5. Stakeholder Assignment. In case a stakeholder accepts the assignment he / she will be marked in green.	77
3.6. Details about the inconsistencies of the current requirements model.	77
3.7. QT plugin visualization of a requirement / issue called "QTBUG-46129"	78
3.8. ECLIPSE plugin presenting a personalized prioritized list of ECLIPSE requirements for a developer of the ECLIPSE IDE.	79
3.9. ECLIPSE plugin settings page including calculated keywords.	79
5.1. One-dimensional 5-star rating interface. Stakeholders express their estimation of the importance of a requirement using a 5-star rating scale. The average of these votes represents the utility value (here: 3.67).	105
5.2. Overview of multidimensional MAUT ratings for a requirement. In this example, three stakeholders evaluated a requirement based on the interest dimensions $\{profit, risk, effort\}$. Given these votes, the utility value can be calculated using Formula 5.1.	106

5.3.	Recommended prioritization of requirements. The utility values (see right side) reflect the evaluated priority of a requirement and are calculated based on the evaluations provided by the stakeholders for the specific requirement. A high utility value of a requirement indicates a high priority which advises decision makers (e.g., requirements managers) to consider the requirement in earlier releases rather than requirements of a lower utility.	107
5.4.	Basic example of a vote-inheritance tree for one interest dimension (or one requirement). In terms of liquid democracy, stakeholders (nodes) can delegate their vote to a different stakeholder (directed edge).	108
5.5.	Argumentation-based rating interface. In addition to the basic MAUT version, stakeholders are enabled to exchange arguments for / against specific requirements. Positive arguments are highlighted in green, neutral arguments in orange, and negative arguments in red. After entering the text for an argument, the system asks for the interest dimension that touches the user's argument.	110
5.6.	Study results of our three UI types showing statistical details about the number of (a) <i>rating evaluations</i> , (b) <i>rating adaptations / interactions</i> , and (c) <i>points achieved by the students</i> (maximum number of points = 30). In all three plots, the marker "x" represents the mean of the distribution and all outliers are shown in the form of the symbol "o". The median values of plots (a) and (b) cannot be identified since they either overlap with the first or the third quartile. The median values of these plots are as follows: (a) 5-star rating: 4, basic MAUT: 4, argumentation-based MAUT: 5; (b) 5-star rating: 5, basic MAUT: 5, argumentation-based MAUT: 10.	114
6.1.	Evaluation of stakeholders in OPENREQ!LIVE. Each stakeholder-assignment is evaluated by two evaluation dimensions (appropriateness and availability). The utility value of an evaluated stakeholder is calculated by using Formula 6.2.	128
8.1.	ECLIPSE plugin for the contributor-specific prioritization of bugs.	145
8.2.	Histograms of the developer activity in different open-source communities (ECLIPSE, MOZILLA, and LIBREOFFICE). The x-axis refers to the number of resolved issues (ascending order) and the y-axis reports the corresponding number of contributors in logarithmic scale.	148
8.3.	Steps performed for preprocessing and feature extraction.	149
8.4.	Precision scores of the different classifiers and the constant predictor baseline. The highest precision rates for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE) were achieved by Random Forest.	154

8.5. Recall rates of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE). The constant predictor baseline represents a very simple "dummy" classifier that always recommends <i>all</i> available issues and therefore achieves the highest possible recall rate.	155
8.6. F1-scores of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, and LIBREOFFICE).	156
9.1. Overview of OPENREQ's requirements engineering approach.	162
9.2. Overview of an example project in OPENREQ!LIVE. Requirements (they consist of a unique ID, title, description, and status) are listed on the page and ordered by a utility value. Each release has constraints such as the deadline of the release and a maximum capacity value (in hours) which limits the possible number of requirements that can be assigned to the release. The OPENREQ services (see Section 9.3.3) have been integrated into the user interface. For example, the red labeled numbers indicate issues (such as requirement duplicates, ambiguities in a requirement's description text, etc.) reported by some of the services.	170
9.3. Argumentation-based rating interface which allows stakeholders to exchange arguments for / against a requirement. Each argument must be assigned to one interest dimension. Negative arguments are highlighted in red, positive arguments in green, and neutral arguments in orange.	171
9.4. ECLIPSE plugin that recommends relevant requirements / issues to the active developer using content-based filtering based on supervised learning.	172

List of Tables

1.1.	Contributions of this thesis with regard to the corresponding research questions. . . .	15
2.1.	A basic group-based ranking scenario. Group members u_i provide ranks for items $t_i \in I$ (alternatively, rankings can be derived by a recommender). Thereafter, an aggregation function such as <i>borda count</i> (BRC) can be used to derive a corresponding ranking for the group. The \surd symbol indicates the recommended item.	23
2.2.	A group-based packaging scenario. Users provide ranks for items t_{ij} (j^{th} item of type i). Thereafter, an aggregation function such as <i>borda count</i> (BRC) can be used for deriving a proposed package (in our case, $\{t_{11}, t_{21}, t_{31}\}$). The \surd symbol indicates the recommended items part of the package.	24
2.3.	Group-based parametrization. Users define preferences with regard to the parameters par_i . Thereafter, an aggregation function such as <i>majority voting</i> (MAJ) can be used for recommending a parametrization (in our case, $\{par_1 = a, par_2 = 1, par_3 = 2\}$). The \surd symbol indicates recommended parameter values.	25
2.4.	A group-based configuration scenario. Users u_i specify their preferences in terms of parameter values. Constraints c_i specify the restrictions, a configuration must take into account. Thereafter, an aggregation function such as <i>least misery</i> (LMS) can be used for deriving a recommended configuration (in our case, $\{par_1 = a, par_2 = 1, par_3 = u, par_4 = 1\}$). The \surd symbol indicates the configuration parameter values recommended to the group.	25
2.5.	A group-based release planning scenario. Users can specify their preferences in terms of assignments of requirements (req_i) to releases. Additionally, constraints c_i specify properties a release plan must take into account. Thereafter, an aggregation function such as <i>least misery</i> can be used for deriving a proposed release plan (in our case, for example, release plan 2). The \surd symbol indicates recommended release plans. . . .	26

2.6. Group-based triage. Users specify their preferences by categorizing requirements (req_i) into a (accept), m (maybe accept), and r (reject). Constraints c_1 and c_2 specify dependencies between requirements, c_3 specifies that two requirements have to be accepted (a). An aggregation function such as <i>least misery</i> (LMS) can be used for deriving a triage solution (in our case, triage 1). The \surd symbol indicates the triage recommended to the group.	27
2.7. Group-based resource balancing. Users specify their preferences with regard to resource assignments in terms of <i>ratings</i> . Constraints c_i specify properties a resource assignment must take into account. <i>Least misery</i> (LMS) denotes the lowest user-specific evaluation of a resource assignment. The \surd symbol indicates the recommended assignment (in our case, assignment 4).	28
2.8. Group-based sequencing. Users specify their preferences in terms of assignments of sequential numbers to items t_i . Additionally, constraints c_i specify properties a sequence must take into account. Here, u_{it_j} is a parameter representing a user's (u_i) assignment of item t_j to a specific sequence position. <i>Least misery</i> (LMS) denotes the number of times, a user preference is neglected by a sequence. Sequences $id = 1$ and $id = 2$ can be regarded as recommendation candidates.	29
2.9. A sequencing scenario where different sequences are explicitly defined, i.e., the choice task is 'reduced' to a ranking scenario. In this example, sequence 1 has the highest <i>average</i> (AVG) value, i.e., it will be recommended first.	30
2.10. Evaluation scheme of polls and questionnaires – persons providing feedback often do not participate in the related decision making process.	30
2.11. A voting process. Each user is allowed to give only one vote – a decision is made on the basis of the ADD aggregation function.	31
3.1. Basic example to demonstrate the potential of content-based recommendation approaches that foster the reuse of requirements in RE. In this example, the requirements 1, 2, and 5 (highlighted in bold) of an existing project A are recommended to be included in a new project which is project B.	46
3.2. Basic example to demonstrate the potential of collaborative filtering approaches in RE. Ann and Chris have rated the same requirements (requirement 2 and 3) in a similar fashion as the active user Bob. Hence, Ann and Chris are considered as similar / neighbor users and requirement 4 (which is highlighted in bold and has been highly rated by them but has not been seen by Bob yet) is recommended to Bob.	48
3.3. Basic example of a recommended release plan (requirements are abbreviated as "Req.") considering requirement- and release-related constraints such as requirement effort, requirement priority, interdependencies between the requirements, and the maximum capacity of the releases	50

3.4.	Three stakeholders ($S=\{Ann, Chris, Susan\}$) evaluated the requirements $\{r_1, r_2, r_3\}$ with regard to the attributes <i>profit</i> , <i>effort</i> , and <i>risk</i> . The calculation of the utility values is based on Formula 3.8. The priority ranking defines the suggested ranking of the requirements based on their determined utility.	53
3.5.	Expertise level $w(s, d)$ of the individual stakeholders in the range between 0.0 (minimum) and 1.0 (maximum) with respect to the interest dimensions profit, effort, and risk.	54
3.6.	Overview of the RE process and its tasks. The table includes references of relevant recommendation approaches used during the different activities of the process. Activities that are not an integral part of RE (such as the design, implementation, and testing of the software as well as the maintenance of the software project), are not shown in this table.	59
3.7.	General overview of the different recommendation approaches and basic criteria to guide the selection of an algorithm.	80
4.1.	Dependencies found by experts and students.	92
4.2.	Scores of the different algorithms (precision [P], recall [R], and f1-score [F1]). The highest scores are highlighted.	97
5.1.	Three stakeholders evaluated the requirements $\{r_1, r_2, r_3\}$ with regard to the dimensions profit (p), effort (e), risk (r). The calculation of the utility values is based on Formula 5.1. The priority ranking defines the suggested ranking of the requirements based on their determined utility.	109
5.2.	Assumed expertise of the stakeholders $\{Ann, Chris, Susan\}$ in the range between 0.0 and 1.0 with regard to the dimensions profit (p), effort (e), and risk (r) (0.0 = very low ... 1.0 = very high).	109
5.3.	Overview of our study results showing general details about the number of requirements, the number of evaluations, and the total number of rating / evaluation interactions with regard to the corresponding UI version (5-star rating, basic MAUT, argumentation-based MAUT).	115
5.4.	Statistical sentiment analysis of the requirement evaluations. Based on the rating value (ranging from 1 to 5) we classified the requirement ratings / evaluations into three different sentiment levels (negative: 1 and 2, neutral: 3, positive: 4 and 5). The dominating sentiment levels for each UI type are highlighted in bold.	116
6.1.	Examples of domain identifiers for rail automation	122
6.2.	Initial assignment of stakeholders to requirements done by requirements manager (RM). The dash symbol ("'-") indicates that the other stakeholders have not made a decision yet.	123
6.3.	State of assignment during assessment phase	123

6.4.	Final state after assessment phase. Consistent assignment of stakeholders to requirements.	124
6.5.	State of assignment with group decision service (GDS) and stakeholder recommendation service (RS1). The recommendation service provides a confidence value which lies in the range between 1 and 10.	125
7.1.	Contribution of requirements $\{r_1, r_2, r_3\}$ to the interest dimensions $\{profit, risk, effort\}$	135
7.2.	Predefined weights for the interest dimensions $D = \{profit, risk, effort\}$	135
7.3.	Ranking of requirements with static weights.	136
7.4.	Contribution of requirements $R = \{r_1, r_2, r_3\}$ to dimensions $D = \{profit, risk, effort\}$ (defined by stakeholders $S = \{s_1, s_2, s_3\}$).	136
7.5.	Preferences of stakeholders $S = \{s_1, s_2, s_3\}$ with regard to the interest dimensions $D = \{profit, risk, effort\}$	137
7.6.	Ranking of requirements with group weights.	137
7.7.	Contribution of requirements (bugs) $R = \{r_1, r_2, r_3\}$ to the interest dimensions $D = \{cc, geritt, blocker, comments\}$	138
7.8.	Expertise of stakeholder s_1 with regard to the requirements $\{r_1, r_2, r_3\}$ determined, for example, on the basis of the similarity between the stakeholder profile and information associated with a requirement.	139
7.9.	Ranking of BUGZILLA bugs with static weights.	139
8.1.	Key characteristics of our issue datasets.	147
8.2.	Hyperparameter optimization using grid search.	152
8.3.	Overview of the number of constructed data records and the vocabulary size of the different datasets.	152
A.1.	Study results of our group recommendation UI types (<i>5-star rating, basic MAUT, argumentation-based MAUT</i>) showing statistical details about the average (avg.) and standard deviation (stdev.) of the evaluation interactions (rating adaptations), the evaluations / ratings, and the points achieved by the students.	189
A.2.	Pairwise comparison of our group recommendation UI types (<i>5-star rating, basic MAUT, argumentation-based MAUT</i>) with respect to statistical differences in the number of ratings / evaluations using a non-parameterized statistical (stat.) significance test (<i>Wilcoxon-Mann-Whitney-Test</i>). The pairwise significance tests were calculated to determine if the diverging observations between the different UI versions are statistically significant (i.e., whether significantly more ratings / evaluations were observed in Y than in X).	190

A.3. Pairwise comparison of our group recommendation UI types (<i>5-star rating, basic MAUT, argumentation-based MAUT</i>) with respect to statistical differences in the number of rating adaptations / interactions using a non-parameterized statistical (stat.) significance test (<i>Wilcoxon-Mann-Whitney-Test</i>). The pairwise significance tests were calculated to determine if the diverging observations between the different UI versions are statistically significant (i.e., whether significantly more rating adaptations / interactions were observed in Y than in X).	190
A.4. Pairwise comparison of our group recommendation UI types (<i>5-star rating, basic MAUT, argumentation-based MAUT</i>) with respect to statistical differences in the number of points achieved by the students using a non-parameterized statistical (stat.) significance test (<i>Wilcoxon-Mann-Whitney-Test</i>). The pairwise significance tests were calculated to determine if the diverging observations between the different UI versions are statistically significant (i.e., whether a significantly higher number of points was observed in Y when compared to X).	190
B.1. Precision scores of the different classifiers and the constant predictor baseline. The highest precision rates for all three datasets (ECLIPSE, MOZILLA, LIBREOFFICE).	191
B.2. Recall rates of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, LIBREOFFICE).	191
B.3. F1-scores of the different classifiers and the constant predictor baseline for all three datasets (ECLIPSE, MOZILLA, LIBREOFFICE) were achieved by Random Forest and are highlighted in bold.	192

Bibliography

- ABAD, Z. S. H., KARRAS, O., GHAZI, P., GLINZ, M., RUHE, G., AND SCHNEIDER, K. 2017. What works better? A study of classifying requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 496–501. (Cited on pages 59 and 60.)
- ABUALHAIJA, S., ARORA, C., SABETZADEH, M., BRIAND, L. C., AND VAZ, E. 2019. A machine learning-based approach for demarcating requirements in textual specifications. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 51–62. (Cited on pages 6, 59, and 60.)
- ACHIMUGU, P., SELAMAT, A., IBRAHIM, R., AND MAHRIN, M. N. 2014. A systematic literature review of software requirements prioritization research. *Information and Software Technology* 56, 6, 568–585. (Cited on pages 132, 133, and 181.)
- ADOMAVICIUS, G., MANOUSELIS, N., AND KWON, Y. 2011. *Multi-Criteria Recommender Systems*. Springer US, Boston, MA, 769–803. (Cited on page 132.)
- ADOMAVICIUS, G. AND TUZHILIN, A. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.* 17, 6 (June), 734–749. (Cited on page 89.)
- AGUILAR, J. A., GARRIGÓS, I., MAZÓN, J.-N., AND ZALDÍVAR, A. 2012. Dealing with dependencies among functional and non-functional requirements for impact analysis in web engineering. ICCSA'12. Springer, Berlin, Heidelberg, 116–131. (Cited on pages 6 and 178.)
- AL-RAWAS, B. AND EASTERBROOK, S. 1996. Communication problems in requirements engineering: A field study. In *Proc. of Conf. on Prof. on Awareness in Software Engineering*. 47–60. (Cited on pages 13 and 109.)
- ALANAZI, E., MOUHOUB, M., AND MOHAMMED, B. 2012. A Preference-Aware Interactive System for Online Shopping. *Computer and Information Science* 5, 6, 33–42. (Cited on page 26.)
- ALDANONDO, M. AND VAREILLES, E. 2008. Configuration for Mass Customization: How to Extend Product Configuration Towards Requirements and Process Configuration. *Journal of Intelligent Manufacturing* 19, 5, 521–535. (Cited on page 21.)
- ALENEZI, M. AND BANITAAN, S. 2013. Bug reports prioritization: Which features and classifier to use? In *2013 12th International Conference on Machine Learning and Applications*. Vol. 2. 112–116. (Cited on pages 102, 132, 133, and 144.)

- ALENEZI, M., BANITAAN, S., AND KENNETH, M. 2013. Efficient bug triaging using text mining. *Proceedings of the 5th International Conference on Computer Engineering and Technology, 2013*. 8, 2185–2190. (Cited on pages 9, 144, and 175.)
- ALGHAZZAWI, D. M., SIDDIQUI, S. T., BOKHARI, M. U., AND HAMATTA, H. S. A. 2014. Selecting appropriate requirements management tool for developing secure enterprises software. *International Journal of Information Technology and Computer Science (IJITCS)* 6, 4, 49–55. (Cited on page 71.)
- AMELLER, D., FARRE, C., FRANCH, X., VALERIO, D., AND CASSARINO, A. 2017. Towards continuous software release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 402–406. (Cited on pages 132 and 144.)
- ANVIK, J., HIEW, L., AND MURPHY, G. C. 2006. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering. ICSE '06*. ACM, New York, NY, USA, 361–370. (Cited on pages 144 and 148.)
- ANVIK, J. AND MURPHY, G. C. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.* 20, 3 (Aug.), 1–35. (Cited on page 148.)
- ARAÚJO, A. A., PAIXAO, M., YELTSIN, I., DANTAS, A., AND SOUZA, J. 2017. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering* 24, 3, 623–671. (Cited on pages 59 and 71.)
- ATAS, M., FELFERNIG, A., STETTINGER, M., AND TRAN, T. 2017. Beyond item recommendation: Using recommendations to stimulate knowledge sharing in group decisions. In *9th International Conference on Social Informatics (SocInfo 2017)*. Oxford, UK, 1–10. (Cited on pages 103 and 141.)
- ATAS, M., SAMER, R., AND FELFERNIG, A. 2018. Automated identification of type-specific dependencies between requirements. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. 688–695. (Cited on pages 59, 69, 90, 91, 102, and 165.)
- ATAS, M., TRAN, T. N. T., SAMER, R., FELFERNIG, A., STETTINGER, M., AND FUCCI, D. 2018. Liquid democracy in group-based configuration. In *Proceedings of the 20th Configuration Workshop, Graz, Austria, September 27-28, 2018*. 93–98. (Cited on pages 52, 54, 64, 103, 104, 105, and 186.)
- AVELINO, G., PASSOS, L., HORA, A., AND VALENTE, M. T. 2016. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 1–10. (Cited on page 9.)
- BAGNALL, A. J., RAYWARD-SMITH, V. J., AND WHITTLEY, I. M. 2001. The next release problem. *Information and Software Technology* 43, 14, 883–890. (Cited on pages 70 and 133.)

-
- BAKER, C., DENG, L., CHAKRABORTY, S., AND DEHLINGER, J. 2019. Automatic multi-class non-functional software requirements classification using neural networks. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2. IEEE Computer Society, 610–615. (Cited on pages 59, 60, and 84.)
- BAKER, P., HARMAN, M., STEINHOFEL, K., AND SKALIOTIS, A. 2006. Search based approaches to component selection and prioritization for the next release problem. In *2006 22nd IEEE International Conference on Software Maintenance*. IEEE Computer Society, 176–185. (Cited on pages 59 and 70.)
- BANO, M. 2015. Addressing the challenges of requirements ambiguity: A review of empirical literature. *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, 21–24. (Cited on pages 67 and 185.)
- BASKIN, J. AND KRISHNAMURTHI, S. 2009. Preference aggregation in group recommender systems for committee decision-making. In *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. ACM, New York, NY, USA, 337–340. (Cited on page 101.)
- BELL, R., KOREN, Y., AND VOLINSKY, C. 2007. The bellkor solution to the netflix prize. Tech. rep., AT&T Research. (Cited on page 3.)
- BERGSTRA, J., BARDENET, R., BENGIO, Y., AND KÉGL, B. 2011. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Curran Associates Inc., USA, 2546–2554. (Cited on page 151.)
- BERGSTRA, J. AND BENGIO, Y. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, 1 (Feb.), 281–305. (Cited on page 157.)
- BERRY, D. M. 2008. Ambiguity in natural language requirements documents. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, B. Paech and C. Martell, Eds. Springer, Berlin, Heidelberg, 1–7. (Cited on page 102.)
- BERRY, D. M. AND KAMSTIES, E. 2004. *Ambiguity in Requirements Specification*. Springer, Boston, MA, 7–44. (Cited on page 6.)
- BHATTACHARYA, P., NEAMTIU, I., AND SHELTON, C. R. 2012. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software* 85, 10 (Oct.), 2275–2292. (Cited on pages 148 and 182.)
- BINKHONAIN, M. AND ZHAO, L. 2019. A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X 1*, 100001. (Cited on pages 59 and 60.)
- BOEHM, B., GRUNBACHER, P., AND BRIGGS, R. O. 2001. Developing groupware for requirements negotiation: lessons learned. *IEEE Software* 18, 3, 46–55. (Cited on page 6.)

- BOEHM, B. W. 1981. *Software Engineering Economics*, 1st ed. Prentice Hall PTR, USA. (Cited on pages 2, 164, and 181.)
- BOKHARI, M. U. AND SIDDIQUI, S. T. 2010. A comparative study of software requirements tools for secure software development. *International Journal of Information Technology (BIJIT)* 2, 2, 1–12. (Cited on page 36.)
- BORATTO, L. 2016. Group recommender systems: State of the art, emerging aspects and techniques, and research challenges. In *Advances in Information Retrieval*, N. Ferro, F. Crestani, M. Moens, J. Mothe, F. Silvestri, G. Di Nunzio, C. Hauff, and G. Silvello, Eds. Springer, 889–892. (Cited on page 101.)
- BURKE, R. 2000. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems* 69, 32, 180–200. (Cited on page 49.)
- BURKE, R. 2002. Hybrid recommender systems: Survey and experiments. *UMUAI Journal* 12, 4, 331–370. (Cited on page 3.)
- BURKE, R. 2007. *Hybrid Web Recommender Systems*. Springer, Berlin, Heidelberg, 377–408. (Cited on page 43.)
- BURKE, R., FELFERNIG, A., AND GOEKER, M. 2011. Recommender systems: An overview. *AI Magazine* 32, 3, 13–18. (Cited on page 49.)
- CABALLERO, A., MUÑOZ, A., SOTO, J., AND BOTÍA, J. A. 2014. Resource assignment in intelligent environments based on similarity, trust and reputation. *J. Ambient Intell. Smart Environ.* 6, 2 (Mar.), 199–214. (Cited on page 102.)
- CAMPOS, P. G., DIEZ, F., AND CANTADOR, I. 2014. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction* 24, 1-2 (Feb.), 67–119. (Cited on page 43.)
- CANT, T., MCCARTHY, J., AND STANLEY, R. 2006. Tools for requirements management: A comparison of telelogic doors and the hive. Tech. rep., Australian Government Department of Defense, Defence Science and Technology Organisation. (Cited on page 1.)
- CARLSHAMRE, P., SANDAHL, K., LINDVALL, M., REGNELL, B., AND NATTOCH DAG, J. 2001. An industrial survey of requirements interdependencies in software product release planning. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*. RE '01. IEEE Computer Society, Washington, DC, USA, 84–91. (Cited on pages 68, 90, and 102.)
- CASAMAYOR, A., GODOY, D., AND CAMPO, M. 2010. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology* 52, 4, 436–445. (Cited on pages 59 and 60.)
- CASTRO-HERRERA, C. AND CLELAND-HUANG, J. 2010. Utilizing recommender systems to support software requirements elicitation. In *Proceedings of the 2nd International Workshop on Rec-*

-
- ommendation Systems for Software Engineering*. RSSE '10. ACM, New York, NY, USA, 6–10. (Cited on pages 59 and 63.)
- CASTRO-HERRERA, C., DUAN, C., CLELAND-HUANG, J., AND MOBASHER, B. 2008. Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes. In *16th IEEE International Requirements Engineering Conference*. 165–168. (Cited on pages 40, 59, 65, and 120.)
- CASTRO-HERRERA, C., DUAN, C., CLELAND-HUANG, J., AND MOBASHER, B. 2009. A recommender system for requirements elicitation in large-scale software projects. In *Proceedings of the 2009 ACM Symposium on Applied Computing*. SAC '09. ACM, New York, NY, USA, 1419–1426. (Cited on pages 4, 40, and 175.)
- CATANIO, J. 2006. Requirements analysis: A review. In *Advances in Systems, Computing Sciences and Software Engineering*, T. Sobh and K. Elleithy, Eds. Springer, Dordrecht, 411–418. (Cited on page 115.)
- CHARNESS, G., GNEEZY, U., AND KUHN, M. A. 2012. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization* 81, 1, 1–8. (Cited on pages 12 and 180.)
- CHEN, L., DE GEMMIS, M., FELFERNIG, A., LOPS, P., RICCI, F., AND SEMERARO, G. 2013. Human decision making and recommender systems. *ACM Trans. Interact. Intell. Syst.* 3, 3 (Oct.). (Cited on page 3.)
- CHITCHYAN, R. AND RASHID, A. 2006. Tracing requirements interdependency semantics. In *Workshop on Early Aspects*. (Cited on pages 90 and 175.)
- CHOU, Y. 2019. *Actionable gamification: Beyond points, badges, and leaderboards*. Packt Publishing Ltd. (Cited on page 86.)
- CLELAND-HUANG, J., BERENBACH, B., CLARK, S., SETTIMI, R., AND ROMANOVA, E. 2007. Best practices for automated traceability. *Computer* 40, 6 (June), 27–35. (Cited on page 68.)
- CLELAND-HUANG, J., CHANG, C. K., AND CHRISTENSEN, M. 2003. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering* 29, 9, 796–810. (Cited on pages 3, 164, and 181.)
- CLELAND-HUANG, J., DUMITRU, H., DUAN, C., AND CASTRO-HERRERA, C. 2009. Automated support for managing feature requests in open forums. *Commun. ACM* 52, 10 (Oct.), 68–74. (Cited on pages 58, 59, and 67.)
- COUGHLAN, J. AND MACREDIE, R. 2002. Effective communication in requirements elicitation: A comparison of methodologies. *Requir. Eng.* 7, 2 (June), 47–60. (Cited on pages 13 and 109.)
- CROSSEN, A., BUDZIK, J., AND HAMMOND, K. J. 2002. Flytrap: Intelligent group music recommendation. In *Proceedings of the 7th International Conference on Intelligent UI*. IUI '02. ACM, New York, NY, USA, 184–185. (Cited on page 102.)

- DALPIAZ, F., DELL'ANNA, D., AYDEMIR, F. B., AND ÇEVİKOL, S. 2019. Requirements classification with interpretable machine learning and dependency parsing. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 142–152. (Cited on pages 59 and 60.)
- DALPIAZ, F., VAN DER SCHALK, I., AND LUCASSEN, G. 2018. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP. In *Requirements Engineering: Foundation for Software Quality*. Springer, 119–135. (Cited on pages 59 and 67.)
- DAMASIOTIS, V., FITSILIS, P., CONSIDINE, P., AND O'KANE, J. 2017. Analysis of software project complexity factors. In *Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences*. ICMSS '17. Association for Computing Machinery, New York, NY, USA, 54–58. (Cited on pages 3 and 36.)
- DANYLENKO, A. AND LÖWE, W. 2012. Context-aware recommender systems for non-functional requirements. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. 80–84. (Cited on pages 59 and 67.)
- DARWISH, N. R. AND MEGAHED, S. 2016. Requirements engineering in scrum framework. *International Journal of Computer Applications* 149, 8 (Sep), 24–29. (Cited on page 2.)
- DAVIS, A., DIESTE, O., HICKEY, A., JURISTO, N., AND MORENO, A. M. 2006. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*. RE '06. IEEE Computer Society, Washington, DC, USA, 176–185. (Cited on page 90.)
- DAVIS, A. M. 2003. The art of requirements triage. *Computer* 36, 3 (Mar.), 42–49. (Cited on pages 4 and 70.)
- DAVIS, A. M. 2005. *Just Enough Requirements Management: Where Software Development Meets Marketing*. Dorset House Publishing Co., Inc., USA. (Cited on pages 2, 36, 160, 164, and 181.)
- DAVIS, J. AND GOADRICH, M. 2006. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. ACM, New York, NY, USA, 233–240. (Cited on page 153.)
- DE SMEDT, T. AND DAELEMANS, W. 2012. Pattern for python. *The Journal of Machine Learning Research* 13, 2063–2067. (Cited on page 149.)
- DEERWESTER, S., DUMAIS, S. T., FURNAS, G. W., LANDAUER, T. K., AND HARSHMAN, R. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41, 6, 391–407. (Cited on pages 69, 95, and 164.)
- DEKHTYAR, A. AND FONG, V. 2017. RE data challenge: Requirements identification with word2vec and tensorflow. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 484–489. (Cited on pages 59, 60, and 84.)

- DEL SAGRADO, J., ÁAGUILA, I. M., AND ORELLANA, F. J. 2011. Requirements interaction in the next release problem. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '11. Association for Computing Machinery, New York, NY, USA, 241–242. (Cited on pages 59 and 70.)
- DENG, L. AND LIU, Y. 2018. *Deep Learning in Natural Language Processing*, 1st ed. Springer. (Cited on page 186.)
- DESHPANDE, G. 2019. Sreyantra: Automated software requirement inter-dependencies elicitation, analysis and learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 186–187. (Cited on pages 90 and 175.)
- DESHPANDE, G., ARORA, C., AND RUHE, G. 2019. Data-driven elicitation and optimization of dependencies between requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 416–421. (Cited on pages 6, 58, 59, 69, 85, and 185.)
- DHINAKARAN, V. T., PULLE, R., AJMERI, N., AND MURUKANNAIAH, P. K. 2018. App review analysis via active learning: Reducing supervision effort without compromising classification accuracy. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 170–181. (Cited on pages 59 and 60.)
- DU, G. AND RUHE, G. 2009. Does explanation improve the acceptance of decision support for product release planning? In *3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 56–68. (Cited on page 141.)
- DUAN, C., LAURENT, P., CLELAND-HUANG, J., AND KWIATKOWSKI, C. 2009. Towards automated requirements prioritization and triage. *Requir. Eng.* 14, 2 (Apr.), 73–89. (Cited on pages 59, 63, and 102.)
- DUMITRU, H., GIBIEC, M., HARIRI, N., CLELAND-HUANG, J., MOBASHER, B., CASTRO-HERRERA, C., AND MIRAKHORLI, M. 2011. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. ACM, New York, NY, USA, 181–190. (Cited on pages 6, 59, and 62.)
- DURME, B. V. AND LALL, A. 2009. Streaming pointwise mutual information. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*. NIPS'09. Curran Assoc. Inc., USA, 1892–1900. (Cited on page 93.)
- DYER, J. 1997. Multi attribute utility theory. *International Series in Operations Research and Management Science* 78, 265–292. (Cited on page 133.)
- EBERT, C. 2014. *Systematisches Requirements Engineering*, 5th ed. dpunkt, Heidelberg, Germany. (Cited on pages 1 and 36.)
- EKSTRAND, M. D., RIEDL, J. T., AND KONSTAN, J. A. 2011. Collaborative filtering recommender systems. *Found. Trends Hum.-Comput. Interact.* 4, 2 (Feb.), 81–173. (Cited on pages 5, 45, and 161.)

- FALKNER, A., FELFERNIG, A., AND HAAG, A. 2011. Recommendation Technologies for Configurable Products. *AI Magazine* 32, 3, 99–108. (Cited on page 26.)
- FALKNER, A., PALOMARES, C., FRANCH, X., SCHENNER, G., AZNAR, P., AND SCHOERGHUBER, A. 2019. Identifying requirements in requests for proposal: A research preview. In *Requirements Engineering: Foundation for Software Quality*. Springer, 176–182. (Cited on pages 59, 60, 162, and 174.)
- FARSHIDI, S., JANSEN, S., JONG, R. D., AND BRINKKEMPER, S. 2018. Multiple criteria decision support in requirements negotiation. In *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track*. 100–107. (Cited on page 6.)
- FELFERNIG, A., ATAS, M., SAMER, R., STETTINGER, M., TRAN, T. N. T., AND REITERER, S. 2018. *Further Choice Scenarios*. Springer, 129–144. (Cited on page 19.)
- FELFERNIG, A., ATAS, M., TRAN, T. T., AND STETTINGER, M. 2016. Towards group-based configuration. In *International Workshop on Configuration 2016 (ConfWS'16)*. 69–72. (Cited on pages 21, 25, 33, and 119.)
- FELFERNIG, A., BAGLEY, C., TIIHONEN, J., WORTLEY, L., AND HOTZ, L. 2014. Chapter 4 - benefits of configuration systems. In *Knowledge-Based Configuration*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann, Boston, 29 – 33. (Cited on page 48.)
- FELFERNIG, A., BORATTO, L., STETTINGER, M., AND TKALČIČ, M. 2018. *Biases in Group Decisions*. Springer International Publishing, 145–155. (Cited on page 173.)
- FELFERNIG, A., BORATTO, L., STETTINGER, M., AND TKALCIC, M. 2018. *Group Recommender Systems – An Introduction*. Springer. (Cited on pages 4, 5, 33, 43, 51, 52, 84, 101, 102, 104, 105, 124, 125, 132, 136, 141, 161, and 165.)
- FELFERNIG, A. AND BURKE, R. 2008. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce*. ICEC '08. Association for Computing Machinery, New York, NY, USA. (Cited on pages 32 and 84.)
- FELFERNIG, A., HOTZ, L., BAGLEY, C., AND TIIHONEN, J. 2014. *Knowledge-based Configuration: From Research to Business Cases*, 1st ed. Elsevier/Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on pages 21, 22, 25, 26, and 119.)
- FELFERNIG, A., JERAN, M., NINAUS, G., REINFRANK, F., AND REITERER, S. 2013. *Toward the Next Generation of Recommender Systems: Applications and Research Challenges*. Springer, Heidelberg, 81–98. (Cited on pages 59 and 66.)
- FELFERNIG, A., JERAN, M., NINAUS, G., REINFRANK, F., REITERER, S., AND STETTINGER, M. 2014. *Basic Approaches in Recommendation Systems*. Springer, Berlin, Heidelberg, 15–37. (Cited on pages 5, 23, 49, and 161.)

-
- FELFERNIG, A. AND NINAUS, G. 2012. Group recommendation algorithms for requirements prioritization. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. 59–62. (Cited on pages 59 and 64.)
- FELFERNIG, A., NINAUS, G., GRABNER, H., REINFRANK, F., WENINGER, L., PAGANO, D., AND MAALEJ, W. 2013. An overview of recommender systems in requirements engineering. *Managing Requirements Knowledge*, 315–332. (Cited on pages 4, 41, 58, 63, 66, 68, 90, 160, and 168.)
- FELFERNIG, A., POLAT-ERDENIZ, S., URAN, C., REITERER, S., ATAS, M., TRAN, T. N. T., AZZONI, P., KIRALY, C., AND DOLUI, K. 2019. An overview of recommender systems in the internet of things. *J. Intell. Inf. Syst.* 52, 2 (Apr.), 285–309. (Cited on pages 80 and 81.)
- FELFERNIG, A., SCHUBERT, M., FRIEDRICH, G., MANDL, M., MAIRITSCH, M., AND TEPPAN, E. 2009. Plausible repairs for inconsistent requirements. In *21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. Pasadena, CA, 791–796. (Cited on page 33.)
- FELFERNIG, A., SCHUBERT, M., MANDL, M., AND GHIRARDINI, P. 2010. Diagnosing inconsistent requirements preferences in distributed software projects. In *Proceedings of the 3rd International Workshop on Social Software Engineering*. 495–502. (Cited on pages 59 and 66.)
- FELFERNIG, A., SCHUBERT, M., AND REITERER, S. 2013. Personalized Diagnosis for Over-Constrained Problems. In *23rd International Conference on Artificial Intelligence (IJCAI 2013)*. Peking, China, 1990–1996. (Cited on page 33.)
- FELFERNIG, A., SCHUBERT, M., AND ZEHENTNER, C. 2012. An efficient diagnosis algorithm for inconsistent constraint sets. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)* 26, 1, 175–184. (Cited on pages 25, 134, 139, and 140.)
- FELFERNIG, A., SPÖCKLBERGER, J., SAMER, R., STETTINGER, M., ATAS, M., TIIHONEN, J., AND RAATIKAINEN, M. 2018. Configuring release plans. In *Proceedings of the 20th Configuration Workshop, Graz, Austria, September 27-28, 2018*. 9–14. (Cited on pages 51, 164, and 168.)
- FELFERNIG, A., STETTINGER, M., ATAS, M., SAMER, R., NERLICH, J., SCHOLZ, S., TIIHONEN, J., AND RAATIKAINEN, M. 2018. Towards utility-based prioritization of requirements in open source environments. In *26th IEEE International Requirements Engineering Conference (RE 2018)*, Banff, AB, Canada, August 20-24, 2018. 406–411. (Cited on pages 59, 64, 102, 131, and 144.)
- FELFERNIG, A., STETTINGER, M., FALKNER, A., ATAS, M., FRANCH, X., AND PALOMARES, C. 2017. OPENREQ: Recommender systems in requirements engineering. In *2nd Workshop on Recommender Systems and Big Data Analytics. RS-BDA'17*. 1–4. (Cited on pages 32, 36, 72, 76, and 120.)
- FELFERNIG, A., TEPPAN, E., AND GULA, B. 2007. Knowledge-based recommender technologies for marketing and sales. *International Journal of Pattern Recognition and Artificial Intelligence* 21, 333–354. (Cited on page 79.)
-

- FELFERNIG, A., ZEHENTNER, C., NINAUS, G., GRABNER, H., MAALEJ, W., PAGANO, D., WENINGER, L., AND REINFRANK, F. 2011. Group decision support for requirements negotiation. In *Proceedings of the 19th International Conference on Advances in User Modeling*. UMAP'11. Springer, Berlin, Heidelberg, 105–116. (Cited on pages 6, 21, and 63.)
- FEMMER, H., FERNÁNDEZ, D. M., WAGNER, S., AND EDER, S. 2017. Rapid quality assurance with requirements smells. *Journal of Systems and Software* 123, 190–213. (Cited on pages 59 and 67.)
- FERBER, S., HAAG, J., AND SAVOLAINEN, J. 2002. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In *Software Product Lines*. Springer, Berlin, Heidelberg, 235–256. (Cited on pages 6, 69, 90, 98, and 178.)
- FERNÁNDEZ, D. 2018. Supporting requirements-engineering research that industry needs: The napire initiative. *IEEE Software* 35, 1 (January), 112–116. (Cited on page 2.)
- FERRARI, A., DELL'ORLETTA, F., SPAGNOLO, G. O., AND GNESI, S. 2014. Measuring and improving the completeness of natural language requirements. In *Proceedings of the 20th International Working Conference on Requirements Engineering: Foundation for Software Quality - Volume 8396*. REFSQ 2014. Springer, Berlin, Heidelberg, 23–38. (Cited on page 6.)
- FERRARI, A. AND ESULI, A. 2019. An NLP approach for cross-domain ambiguity detection in requirements engineering. *Automated Software Engineering* 26, 3, 559–598. (Cited on pages 59 and 67.)
- FERRARI, A. AND GNESI, S. 2012. Using collective intelligence to detect pragmatic ambiguities. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. 191–200. (Cited on pages 59 and 67.)
- FERRARI, A., GORI, G., ROSADINI, B., TROTTA, I., BACHERINI, S., FANTECHI, A., AND GNESI, S. 2018. Detecting requirements defects with NLP patterns: an industrial experience in the railway domain. *Empirical Software Engineering* 23, 6, 3684–3733. (Cited on pages 59 and 67.)
- FERRARI, A., SPAGNOLO, G. O., AND GNESI, S. 2017. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 502–505. (Cited on page 85.)
- FERRARI, A., SPOLETINI, P., AND GNESI, S. 2016. Ambiguity and tacit knowledge in requirements elicitation interviews. *Requirements Engineering* 21, 3, 333–355. (Cited on page 67.)
- FINKELSTEIN, A., HARMAN, M., MANSOURI, S. A., REN, J., AND ZHANG, Y. 2009. A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. *Requirements Engineering* 14, 4, 231–245. (Cited on pages 59 and 70.)
- FIRESMITH, D. 2004. Prioritizing requirements. *Journal of Object Technology* 3, 35–48. (Cited on page 181.)

- FIRESMITH, D. 2005. Are your requirements complete? *Journal of Object Technology* 4, 27–44. (Cited on page 8.)
- FITZGERALD, C., LETIER, E., AND FINKELSTEIN, A. 2011. Early failure prediction in feature request management systems. In *2011 IEEE 19th International Requirements Engineering Conference*. 229–238. (Cited on pages 6, 59, and 67.)
- FUCCI, D., PALOMARES, C., FRANCH, X., COSTAL, D., RAATIKAINEN, M., STETTINGER, M., KURTANOVIĆ, Z., KOJO, T., KOENIG, L., FALKNER, A., SCHENNER, G., BRASCA, F., MÄNNISTÖ, T., FELFERNIG, A., AND MAALEJ, W. 2018. Needs and challenges for a platform to support large-scale requirements engineering: A multiple-case study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '18. ACM, New York, NY, USA, 19:1–19:10. (Cited on pages 3, 36, 37, and 40.)
- GARCIA, I., SEBASTIA, L., ONAINDIA, E., AND GUZMAN, C. 2009. A group recommender system for tourist activities. In *Proceedings of the 10th International Conference on E-Commerce and Web Technologies*. EC-Web 2009. Springer, Berlin, Heidelberg, 26–37. (Cited on page 102.)
- GARTNER RESEARCH, R. 2014. Hype cycle for application development. <https://www.gartner.com/en/documents/2810920>. [Online; accessed 2021-01-06]. (Cited on page 2.)
- GARTNER RESEARCH, S. 2020. Information technology spending on enterprise software worldwide. <https://www.statista.com/statistics/203428/total-enterprise-software-revenue-forecast/>. [Online; accessed 2021-01-06]. (Cited on page 1.)
- GENA, C., BROGI, R., CENA, F., AND VERNERO, F. 2011. The impact of rating scales on user's rating behavior. In *User Modeling, Adaption and Personalization*, J. Konstan, R. Conejo, J. Marzo, and N. Oliver, Eds. Springer, Berlin, Heidelberg, 123–134. (Cited on page 117.)
- GERVASI, V., FERRARI, A., ZOWGHI, D., AND SPOLETINI, P. 2019. *Ambiguity in Requirements Engineering: Towards a Unifying Framework*. Springer, 191–210. (Cited on page 67.)
- GLEICH, B., CREIGHTON, O., AND KOF, L. 2010. Ambiguity detection: Towards a tool explaining ambiguity sources. In *Requirements Engineering: Foundation for Software Quality, 16th International Working Conference, REFSQ 2010, Essen, Germany, June 30 - July 2, 2010. Proceedings*. Lecture Notes in Computer Science, vol. 6182. Springer, 218–232. (Cited on pages 59 and 67.)
- GOLDBERG, D., NICHOLS, D., OKI, B., AND TERRY, D. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (Dec.), 61–70. (Cited on pages 5, 45, and 161.)
- GOLDIN, L. AND BERRY, D. M. 2015. Reuse of requirements reduced time to market at one industrial shop: A case study. *Requirements Engineering* 20, 1 (Mar.), 23–44. (Cited on pages 59 and 62.)
- GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. 2016. *Deep Learning*. MIT Press. <https://www.deeplearningbook.org>. (Cited on pages 4, 82, and 84.)

- GREER, D. AND RUHE, G. 2004. Software release planning: an evolutionary and iterative approach. *Information and Software Technology* 46, 4, 243–253. (Cited on pages 59, 70, and 115.)
- GREITEMEYER, T. AND SCHULZ-HARDT, S. 2003. Preference-consistent evaluation of information in the hidden profile paradigm: Beyond group-level explanations for the dominance of shared information in group decisions. *Journal of personality and social psychology* 84, 322–39. (Cited on pages 8, 109, and 181.)
- GROEN, E. C., DOERR, J., AND ADAM, S. 2015. Towards crowd-based requirements engineering a research preview. In *Requirements Engineering: Foundation for Software Quality*. Springer, 247–253. (Cited on pages 59 and 61.)
- GUPTA, A. AND DERAMAN, A. 2019. A framework for software requirement ambiguity avoidance. *International Journal of Electrical and Computer Engineering* 9, 5436–5445. (Cited on page 67.)
- GUZMAN, E., IBRAHIM, M., AND GLINZ, M. 2017. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 11–20. (Cited on pages 59 and 61.)
- HARIRI, N., CASTRO-HERRERA, C., CLELAND-HUANG, J., AND MOBASHER, B. 2014. *Recommendation Systems in Requirements Discovery*. Springer, Berlin, Heidelberg, 455–476. (Cited on page 40.)
- HARMAN, M., MCMINN, P., DE SOUZA, J. T., AND YOO, S. 2012. *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*. Springer, Berlin, Heidelberg, 1–59. (Cited on page 70.)
- HART, A. 2001. Mann-whitney test is not just a test of medians: differences in spread can be important. *BMJ* 323, 7309, 391–393. (Cited on page 189.)
- HARUNA, K., ISMAIL, M. A., SUHENDROYONO, S., DAMIASIH, D., PIEREWAN, A., CHIROMA, H., AND HERAWAN, T. 2017. Context-aware recommender system: A review of recent developmental process and future research direction. *Applied Sciences* 7, 1211. (Cited on page 43.)
- HAUGEN, N. 2006. An Empirical Study of Using Planning Poker for User Story Estimation. In *AGILE 2006*. 23–34. (Cited on page 32.)
- HELMING, J., ARNDT, H., HODAIE, Z., KOEGEL, M., AND NARAYAN, N. 2010. Automatic assignment of work items. *ENASE 2010 - Proceedings of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering*, 149–158. (Cited on page 148.)
- HOFFMANN, M., KUHN, N., WEBER, M., AND BITTNER, M. 2004. Requirements for requirements management tools. In *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004*. 301–308. (Cited on pages 1 and 71.)
- HOFMANN, H. F. AND LEHNER, F. 2001. Requirements engineering as a success factor in software projects. *IEEE Softw.* 18, 4 (July), 58–66. (Cited on pages 35, 36, and 120.)

- HU, Y., KOREN, Y., AND VOLINSKY, C. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. 263–272. (Cited on page 48.)
- HUANG, S. 2011. Designing utility-based recommender systems for e-commerce: Evaluation of preference elicitation methods. *Electronic Commerce Research and Applications* 10, 4, 398–407. (Cited on page 132.)
- HUJAINAH, F., BAKAR, R. B. A., ABDULGABBER, M. A., AND ZAMLI, K. Z. 2018. Software requirements prioritisation: A systematic literature review on significance, stakeholders, techniques and challenges. *IEEE Access* 6, 71497–71523. (Cited on page 9.)
- IDRISSI, N. AND ZELLOU, A. 2020. A systematic literature review of sparsity issues in recommender systems. *Social Network Analysis and Mining* 10, 1, 15. (Cited on page 82.)
- IKONOMAKIS, E., KOTSIANTIS, S., AND TAMPAKAS, V. 2005. Text classification using machine learning techniques. In *WSEAS Transactions on Computers*. Vol. 4. 966–974. (Cited on page 127.)
- ISINKAYE, F., FOLAJIMI, Y., AND OJOKOH, B. 2015. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal* 16, 3, 261 – 273. (Cited on page 3.)
- IVAN, G., PACHECO, C., CALVO-MANZANO, J., AND ARCILLA, M. 2016. Reusing functional software requirements in small-sized software enterprises: A model oriented to the catalog of requirements. *Requirements Engineering* 22. (Cited on pages 6, 59, and 62.)
- IYER, J. AND RICHARDS, D. 2004. Evaluation framework for tools that manage requirements inconsistency. In *AWRE'04 9th Australian Workshop on Requirements Engineering*. (Cited on page 66.)
- JAMESON, A., BALDES, S., AND KLEINBAUER, T. 2004. Two methods for enhancing mutual awareness in a group recommender system. In *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. Association for Computing Machinery, New York, NY, USA, 447–449. (Cited on pages 21 and 102.)
- JAMESON, A. AND SMYTH, B. 2007. *Recommendation to Groups*. Springer, Berlin, Heidelberg, 596–627. (Cited on page 124.)
- JANIS, I. L. 1982. *Groupthink: Psychological Studies of Policy Decisions and Fiascoes*. (Cited on page 8.)
- JANNACH, D., ZANKER, M., FELFERNIG, A., AND FRIEDRICH, G. 2010. *Recommender Systems: An Introduction*, 1st ed. Cambridge University Press, New York, NY, USA. (Cited on pages 43, 127, 144, and 146.)
- JOHANN, T. AND MAALEJ, W. 2015. Democratic mass participation of users in Requirements Engineering? In *23rd International Requirements Engineering Conference (RE)*. Ottawa, ON, Canada, 256–261. (Cited on pages 4, 54, 64, 103, 104, 166, and 186.)

- JOHANN, T., STANIK, C., BAHNEMIRI, A. M. A., AND MAALEJ, W. 2017. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 21–30. (Cited on pages 59 and 61.)
- JUNKER, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In *19th National Conference on AI (AAAI04)*. San Jose, CA, 167–172. (Cited on page 140.)
- KAHNG, A., MACKENZIE, S., AND PROCACCIA, A. D. 2018. Liquid democracy: An algorithmic perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1095–1102. (Cited on page 129.)
- KAMSTIES, E. 2005. *Understanding Ambiguity in Requirements Engineering*. Springer, Berlin, Heidelberg, 245–266. (Cited on page 185.)
- KANCHEV, G. M., MURUKANNAIAH, P. K., CHOPRA, A. K., AND SAWYER, P. 2017a. Canary: An interactive and query-based approach to extract requirements from online forums. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 470–471. (Cited on pages 59 and 61.)
- KANCHEV, G. M., MURUKANNAIAH, P. K., CHOPRA, A. K., AND SAWYER, P. 2017b. Canary: Extracting requirements-related information from online discussions. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 31–40. (Cited on pages 59 and 61.)
- KARLSSON, J. 1996. Software requirements prioritizing. In *Proceedings of the Second International Conference on Requirements Engineering*. 110–116. (Cited on pages 59 and 63.)
- KARLSSON, J. AND RYAN, K. 1997. A cost-value approach for prioritizing requirements. *IEEE Software* 14, 5, 67–74. (Cited on page 133.)
- KEENAN, E., CZAUDERNA, A., LEACH, G., CLELAND-HUANG, J., SHIN, Y., MORITZ, E., GETHERS, M., POSHYVANYK, D., MALETIC, J., HAYES, J. H., DEKHTYAR, A., MANUKIAN, D., HOSSEIN, S., AND HEARN, D. 2012. Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *2012 34th International Conference on Software Engineering (ICSE)*. 1375–1378. (Cited on page 68.)
- KHAN, J. A., LIU, L., AND WEN, L. 2020. Requirements knowledge acquisition from online user forums. *IET Software* 14, 3, 242–253. (Cited on pages 59 and 61.)
- KHAN, J. A., XIE, Y., LIU, L., AND WEN, L. 2019. Analysis of requirements-related arguments in user forums. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 63–74. (Cited on pages 59 and 61.)
- KIFETEW, F., MUNANTE, D., PERINI, A., SUSI, A., SIENA, A., AND BUSETTA, P. 2017. Dmgame: A gamified collaborative requirements prioritisation tool. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 468–469. (Cited on pages 59 and 65.)

- KIFETEW, F., SUSI, A., MUTANTE, D., PERINI, A., SIENA, A., AND BUSETTA, P. 2017. Towards multi-decision-maker requirements prioritisation via multi-objective optimisation. In *Forum and Doctoral Consortium Papers Presented at the 29th International Conference on Advanced Information Systems Engineering (CAiSE'17)*. Essen, Germany, 137–144. (Cited on pages 132 and 133.)
- KITCHENHAM, B. AND CHARTERS, S. 2007. Guidelines for performing systematic literature reviews in software engineering. Tech. rep., Keele University and Durham University Joint Report. (Cited on page 38.)
- KOREN, Y., BELL, R., AND VOLINSKY, C. 2009. Matrix factorization techniques for recommender systems. *IEEE Computer* 42, 8, 30–37. (Cited on page 4.)
- KORENIUS, T., LAURIKALA, J., JÄRVELIN, K., AND JUHOLA, M. 2004. Stemming and lemmatization in the clustering of finnish text documents. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management. CIKM '04*. ACM, New York, NY, USA, 625–633. (Cited on page 149.)
- KOTKOV, D., WANG, S., AND VEIJALAINEN, J. 2016. A survey of serendipity in recommender systems. *Know.-Based Syst. III*, C (Nov.), 180–192. (Cited on pages 48 and 83.)
- KRASNER, H. 2018. The cost of poor quality software in the us: A 2018 report. Tech. rep., CISQ Consortium for IT Software Quality. (Cited on page 36.)
- KURTANOVIĆ, Z. AND MAALEJ, W. 2017. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 490–495. (Cited on pages 59 and 60.)
- LANDAUER, T. K., FOLTZ, P. W., AND LAHAM, D. 1998. An introduction to latent semantic analysis. *Discourse Processes* 25, 2-3, 259–284. (Cited on pages 130 and 164.)
- LAU, J. H. AND BALDWIN, T. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the 1st Workshop on Representation Learning for NLP*. Association for Computational Linguistics, Berlin, Germany, 78–86. (Cited on page 130.)
- LAURENT, P., CLELAND-HUANG, J., AND DUAN, C. 2007. Towards automated requirements triage. In *15th International Requirements Engineering Conference*. Delhi, India, 131–140. (Cited on page 138.)
- LEFFINGWELL, D. 1997. Calculating the return on investment from more effective requirements management. *American Programmer* 10, 4, 13–16. (Cited on pages 6, 90, 120, and 160.)
- LEHTOLA, L., KAUPPINEN, M., AND KUJALA, S. 2004. Requirements prioritization – challenges in practice. *International Conference on Product Focused Software Process Improvement (PROFES 2004)*, 497–508. (Cited on pages 131 and 143.)
- LEITNER, G., FERCHER, A., FELFERNIG, A., ISAK, K., ERDENIZ, S. P., AKCAY, A., AND JERAN, M. 2016. Recommending and Configuring Smart Home Installations. In *International Workshop on Configuration 2016 (ConfWS'16)*. 17–22. (Cited on page 21.)

- LEVIN, J. AND NALEBUFF, B. 1995. An Introduction to Vote-Counting Schemes. *Journal of Economic Perspectives* 9, 1, 3–26. (Cited on page 23.)
- LI, C., VAN DEN AKKER, M., BRINKKEMPER, S., AND DIEPEN, G. 2010. An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering* 15, 4, 375–396. (Cited on pages 59 and 70.)
- LI, J., JEFFERY, R., FUNG, K., ZHU, L., WANG, Q., ZHANG, H., AND XU, X. 2012. A business process-driven approach for requirements dependency analysis. In *Business Process Management*, A. Barros, A. Gal, and E. Kindler, Eds. Springer, Berlin, Heidelberg, 200–215. (Cited on page 164.)
- LI, L., HARMAN, M., LETIER, E., AND ZHANG, Y. 2014. Robust next release problem: Handling uncertainty during optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO '14. Association for Computing Machinery, New York, NY, USA, 1247–1254. (Cited on pages 59 and 71.)
- LIM, S. L., QUERCIA, D., AND FINKELSTEIN, A. 2010. 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 - Volume 1*. ICSE '10. ACM, New York, NY, USA, 295–304. (Cited on pages 9, 59, 62, 63, 120, 166, and 182.)
- LINSBAUER, L., FISCHER, S., LOPEZ-HERREJON, R. E., AND EGYED, A. 2015. Using traceability for incremental construction and evolution of software product portfolios. In *Proceedings of the 8th International Symposium on Software and Systems Traceability*. SST '15. IEEE Press, 57–60. (Cited on pages 59 and 68.)
- LINSBAUER, L., LOPEZ-HERREJON, E. R., AND EGYED, A. 2013. Recovering traceability between features and code in product variants. In *Proceedings of the 17th International Software Product Line Conference*. SPLC '13. Association for Computing Machinery, New York, NY, USA, 131–140. (Cited on pages 59 and 68.)
- LIU, T., WANG, Z., TANG, J., YANG, S., HUANG, G., AND LIU, Z. 2019. Recommender systems with heterogeneous side information. In *The World Wide Web Conference*. WWW '19. Association for Computing Machinery, New York, NY, USA, 3027–3033. (Cited on page 43.)
- LOMBRISER, P., DALPIAZ, F., LUCASSEN, G., AND BRINKKEMPER, S. 2016. Gamified requirements engineering: Model and experimentation. In *Requirements Engineering: Foundation for Software Quality*. Springer, 171–187. (Cited on page 86.)
- LU, M. AND LIANG, P. 2017. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. EASE'17. Association for Computing Machinery, New York, NY, USA, 344–353. (Cited on pages 59 and 60.)

- LÜDERS, C. M., RAATIKAINEN, M., MOTGER, J., AND MAALEJ, W. 2019. OPENREQ issue link map: A tool to visualize issue links in jira. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE Computer Society, 492–493. (Cited on page 76.)
- MAALEJ, W. AND THURIMELLA, A. K. 2009. Towards a research agenda for recommendation systems in requirements engineering. In *2009 Second International Workshop on Managing Requirements Knowledge*. 32–39. (Cited on page 41.)
- MAGUIRE, M. AND BEVAN, N. 2002. User requirements analysis: A review of supporting methods. In *Proceedings of the IFIP 17th World Computer Congress - TC13 Stream on Usability: Gaining a Competitive Edge*. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 133–148. (Cited on page 115.)
- MAHMOOD, T. AND RICCI, F. 2009. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM Conference on Hypertext and Hypermedia*. HT '09. ACM, New York, NY, USA, 73–82. (Cited on page 43.)
- MAHMOUD, A. 2015. An information theoretic approach for extracting and tracing non-functional requirements. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. IEEE Computer Society, 36–45. (Cited on pages 59 and 60.)
- MAHMOUD, A. AND NIU, N. 2010. Using semantics-enabled information retrieval in requirements tracing: An ongoing experimental investigation. In *2010 IEEE 34th Annual Computer Software and Applications Conference*. IEEE Computer Society, 246–247. (Cited on page 68.)
- MAHMOUD, A. AND NIU, N. 2011. Tracter: A tool for candidate traceability link clustering. In *2011 IEEE 19th International Requirements Engineering Conference*. IEEE Computer Society, 335–336. (Cited on pages 59 and 68.)
- MANN, H. B. AND WHITNEY, D. R. 1947. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.* 18, 1 (03), 50–60. (Cited on page 189.)
- MAQSOOD, T., FINEGAN, A., AND WALKER, D. 2004. Biases and heuristics in judgment and decision making: The dark side of tacit knowledge. In *Issues in Informing Science and Information Technology I*. 295–301. (Cited on page 63.)
- MASSEY, A. K., RUTLEDGE, R. L., ANTÓN, A. I., AND SWIRE, P. P. 2014. Identifying and classifying ambiguity for regulatory requirements. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE Computer Society, 83–92. (Cited on page 67.)
- MASTHOFF, J. 2004. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction (UMUAI)* 14, 1, 37–85. (Cited on pages 29 and 102.)
- MASTHOFF, J. 2011. Group recommender systems. *Recommender Systems Handbook*, 677–702. (Cited on page 125.)

- MASTHOFF, J. 2015. Group Recommender Systems: Aggregation, Satisfaction and Group Attributes. *Recommender Systems Handbook*, 743–776. (Cited on pages 5, 43, 51, 83, 101, 102, 103, 105, and 161.)
- MENZIES, T. AND MARCUS, A. 2008. Automated severity assessment of software defect reports. In *2008 IEEE International Conference on Software Maintenance*. 346–355. (Cited on page 144.)
- MEZEI, Z. AND EICKHOFF, C. 2017. Evaluating music recommender systems for groups. In *2017 Workshop on Value-Aware and Multistakeholder Recommendation (VAMS 2017)*. (Cited on page 102.)
- MEZGHANI, M., KANG, J., KANG, E., AND SEDES, F. 2019. Clustering for traceability managing in system specifications. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 257–264. (Cited on pages 59 and 68.)
- MEZGHANI, M., KANG, J., AND SÈDES, F. 2018. Industrial requirements classification for redundancy and inconsistency detection in semios. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 297–303. (Cited on pages 59 and 68.)
- MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G., AND DEAN, J. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Curran Associates Inc., USA, 3111–3119. (Cited on page 130.)
- MOBASHER, B. AND CLELAND-HUANG, J. 2011. Recommender systems in requirements engineering. *AI Magazine* 32, 3, 81–89. (Cited on pages 4, 6, 9, 36, 37, 40, 59, 65, 82, 90, 102, 120, 160, and 175.)
- MOJZISCH, A. AND SCHULZ-HARDT, S. 2010. Knowing others' preferences degrades the quality of group decisions. *Journal of Personality and Social Psychology* 98, 5, 794–808. (Cited on pages 109, 111, and 180.)
- MOUGOUEI, D. 2016. Factoring requirement dependencies in software requirement selection using graphs and integer programming. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE 2016. Association for Computing Machinery, New York, NY, USA, 884–887. (Cited on pages 59 and 70.)
- MOUGOUEI, D. AND POWERS, D. M. W. 2020. Dependency-aware software release planning through mining user preferences. *Soft Computing* 24, 15, 11673–11693. (Cited on pages 59 and 70.)
- MURPHY, J., HOFACKER, C. F., AND MIZERSKI, R. 2006. Primacy and recency effects on clicking behavior. *Journal of Computer-Mediated Communication* 11, 2, 522–535. (Cited on page 91.)
- NATT OCH DAG, J., REGNELL, B., CARLSHAMRE, P., ANDERSSON, M., AND KARLSSON, J. 2002. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering* 7, 1, 20–33. (Cited on page 68.)

- NAVARRO-ALMANZA, R., JUAREZ-RAMIREZ, R., AND LICEA, G. 2017. Towards supporting software engineering using deep learning: A case of software requirements classification. In *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 116–120. (Cited on pages 59, 60, and 84.)
- NGO-THE, A. AND RUHE, G. 2008. A systematic approach for solving the wicked problem of software release planning. *Soft Computing* 12, 1, 95–108. (Cited on pages 59 and 71.)
- NGUYEN, T. N. AND RICCI, F. 2017. A chat-based group recommender system for tourism. In *Information and Communication Technologies in Tourism 2017*, R. Schegg and B. Stangl, Eds. Springer International Publishing, 17–30. (Cited on page 32.)
- NINAUS, G. 2012. Using group recommendation heuristics for the prioritization of requirements. In *Proceedings of the 6th ACM Conference on Recommender Systems*. ACM, 329–332. (Cited on pages 6, 102, 103, and 105.)
- NINAUS, G. 2016. Recommendation technologies in requirements engineering. Ph.D. thesis, Graz University of Technology. (Cited on page 36.)
- NINAUS, G., FELFERNIG, A., STETTINGER, M., REITERER, S., LEITNER, G., WENINGER, L., AND SCHANIL, W. 2014. Intellireq: Intelligent techniques for software requirements engineering. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. ECAI'14. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1161–1166. (Cited on pages 21, 26, 49, 52, 59, 64, 71, 72, 73, 102, 103, 104, 105, 144, 160, 163, 165, 168, 175, and 193.)
- NINAUS, G., REINFRANK, F., STETTINGER, M., AND FELFERNIG, A. 2014. Content-based recommendation techniques for requirements engineering. In *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. 27–34. (Cited on pages 4, 6, 40, 41, 59, 68, 90, and 91.)
- NIU, N., BHOWMIK, T., LIU, H., AND NIU, Z. 2014. Traceability-enabled refactoring for managing just-in-time requirements. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 133–142. (Cited on pages 59 and 68.)
- NUSEIBEH, B. AND EASTERBROOK, S. 2000. Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Association for Computing Machinery, New York, NY, USA, 35–46. (Cited on pages 58 and 60.)
- ORIOLE, M., STADE, M., FOTROUSI, F., NADAL, S., VARGA, J., SEYFF, N., ABELLO, A., FRANCH, X., MARCO, J., AND SCHMIDT, O. 2018. Fame: Supporting continuous requirements elicitation by combining user feedback and monitoring. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 217–227. (Cited on pages 59 and 61.)
- OTTER, D. W., MEDINA, J. R., AND KALITA, J. K. 2020. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21. (Cited on page 186.)

- PACHECO, C. AND GARCIA, I. 2008. Stakeholder identification methods in software requirements: Empirical findings derived from a systematic review. In *The Third International Conference on Software Engineering Advances*. 472–477. (Cited on page 182.)
- PACHECO, C. AND TOVAR, E. 2007. Stakeholder identification as an issue in the improvement of software requirements quality. In *Advanced Information Systems Engineering*, J. Krogstie, A. Opdahl, and G. Sindre, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 370–380. (Cited on page 182.)
- PAGANO, D. AND BRUEGGE, B. 2013. User involvement in software evolution practice: A case study. In *2013 35th International Conference on Software Engineering (ICSE)*. 953–962. (Cited on page 61.)
- PALOMARES, C., FRANCH, X., AND FUCCI, D. 2018. Personal recommendations in requirements engineering: THE OPENREQ APPROACH. In *Requirements Engineering: Foundation for Software Quality*, E. Kamsties, J. Horkoff, and F. Dalpiaz, Eds. Springer, 297–304. (Cited on pages 4, 40, 41, 42, 72, 76, and 166.)
- PARRA, E., DE LA VARA, J. L., AND ALONSO, L. 2018. Poster: Analysis of requirements quality evolution. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. 199–201. (Cited on pages 59 and 67.)
- PATHAK, B., GARFINKEL, R., GOPAL, R., VENKATESAN, R., AND YIN, F. 2010. Empirical analysis of the impact of recommender systems on sales. *J. Manage. Inf. Syst.* 27, 2 (Oct.), 159–188. (Cited on page 3.)
- PAZZANI, M. J. AND BILLSUS, D. 1997. Learning and revising user profiles: The identification of interesting web sites. *Mach. Learn.* 27, 3 (June), 313–331. (Cited on page 43.)
- PAZZANI, M. J. AND BILLSUS, D. 2007. *Content-Based Recommendation Systems*. Springer, Berlin, Heidelberg, 325–341. (Cited on pages 4, 43, 44, and 161.)
- PERINI, A., SUSI, A., AND AVESANI, P. 2013. A machine learning approach to software requirements prioritization. *IEEE Transactions on Software Engineering* 39, 4, 445–461. (Cited on pages 59, 63, and 133.)
- PERRONE, M. P., KHAN, H., KIM, C., KYRILLIDIS, A., QUINN, J., AND SALAPURA, V. 2019. Optimal mini-batch size selection for fast gradient descent. Tech. rep., IBM T.J. Watson Research Center. (Cited on page 81.)
- PINTO, J. K. AND MANTEL, S. J. 1990. The causes of project failure. *IEEE Transactions on Engineering Management* 37, 4, 269–276. (Cited on page 2.)
- PITANGUEIRA, A. M. 2015. Incorporating preferences from multiple stakeholders in software requirements selection an interactive search-based approach. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. 382–387. (Cited on pages 59 and 71.)

-
- PITANGUEIRA, A. M., TONELLA, P., SUSI, A., MACIEL, R. S., AND BARROS, M. 2016. Risk-aware multi-stakeholder next release planning using multi-objective optimization. In *Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality - Volume 9619*. REFSQ 2016. Springer, Berlin, Heidelberg, 3–18. (Cited on pages 59 and 70.)
- PITANGUEIRA, A. M., TONELLA, P., SUSI, A., MACIEL, R. S. P., AND BARROS, M. 2017. Minimizing the stakeholder dissatisfaction risk in requirement selection for next release planning. *Inf. Softw. Technol.* 87, 104–118. (Cited on pages 59 and 71.)
- POLAT-ERDENIZ, S., FELFERNIG, A., AND ATAS, M. 2017. Cluster-specific Heuristics for Constraint Solving. In *International Conference on Industrial, Engineering, Other Applications of Applied Intelligent Systems (IEA/AIE)*. Arras, France, 21–30. (Cited on pages 26 and 33.)
- PRZEPIORA, M., KARIMPOUR, R., AND RUHE, G. 2012. A hybrid release planning method and its empirical justification. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '12. ACM, New York, NY, USA, 115–118. (Cited on pages 59 and 70.)
- QADDOURA, R., ABU-SRHAN, A., QASEM, M. H., AND HUDAIB, A. 2017. Requirements prioritization techniques review and analysis. In *2017 International Conference on New Trends in Computing Sciences (ICTCS)*. 258–263. (Cited on pages 7 and 64.)
- QI, S., MAMOULIS, N., PITOURA, E., AND TSAPARAS, P. 2016. Recommending Packages to Groups. In *16th International Conference on Data Mining*. IEEE, 449–458. (Cited on pages 20 and 24.)
- QI, S., MAMOULIS, N., PITOURA, E., AND TSAPARAS, P. 2017. Recommending Packages with Validity Constraints to Groups of Users. *Knowledge and Information Systems*, 1–30. (Cited on pages 20 and 24.)
- QUADRANA, M., CREMONESI, P., AND JANNACH, D. 2018. Sequence-aware recommender systems. *ACM Comput. Surv.* 51, 4 (July). (Cited on page 177.)
- RAATIKAINEN, M., TIHONEN, J., MÄNNISTÖ, T., FELFERNIG, A., STETTINGER, M., AND SAMER, R. 2018. Using a feature model configurator for release planning. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 2*. SPLC '18. ACM, New York, NY, USA, 29–33. (Cited on pages 59, 70, and 168.)
- REGNELL, B., PAECH, B., AURUM, A., WOHLIN, C., DUTOIT, A., AND DAG, J. N. O. 2001. 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)*. 49–52. (Cited on page 8.)
- REITER, R. 1987. A theory of diagnosis from first principles. *AI Journal* 23, 1, 57–95. (Cited on pages 134 and 140.)
-

- RESNICK, P. AND VARIAN, H. R. 1997. Recommender systems. *Commun. ACM* 40, 3 (Mar.), 56–58. (Cited on pages 89 and 101.)
- RICCI, F., ROKACH, L., SHAPIRA, B., AND KANTOR, P. 2010. *Recommender Systems Handbook*, 1st ed. Springer, New York, NY, USA. (Cited on pages 3, 43, and 101.)
- RIEGEL, N. AND DOERR, J. 2015. A systematic literature review of requirements prioritization criteria. In *Requirements Engineering: Foundation for Software Quality*. Springer, 300–317. (Cited on page 64.)
- ROBILLARD, M., WALKER, R., AND ZIMMERMANN, T. 2010. Recommendation Systems for Software Engineering. *IEEE Software* 27, 4, 80–86. (Cited on page 102.)
- ROSA DINI, B., FERRARI, A., GORI, G., FANTECHI, A., GNESI, S., TROTTA, I., AND BACHERINI, S. 2017. Using NLP to detect requirements defects: An industrial experience in the railway domain. In *Requirements Engineering: Foundation for Software Quality*. Springer, 344–360. (Cited on pages 59 and 67.)
- RUHE, G. 2010. *Product Release Planning - Methods, Tools and Applications*. CRC Press. (Cited on pages 6, 90, 115, and 164.)
- RUHE, G., EBERLEIN, A., AND PFAHL, D. 2002. Quantitative WinWin: A new method for decision support in requirements negotiation. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*. SEKE '02. Association for Computing Machinery, New York, NY, USA, 159–166. (Cited on pages 59 and 63.)
- RUHE, G. AND SALIU, M. O. 2005. The art and science of software release planning. *IEEE Software* 22, 6 (Nov.), 47–53. (Cited on pages 132 and 144.)
- RYAN, K. 1993. The role of natural language in requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*. 240–242. (Cited on page 127.)
- SAATY, T. AND VARGAS, L. 2000. *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Springer. (Cited on page 133.)
- SABRIYE, A. O. J. AND ZAINON, W. M. N. W. 2017. A framework for detecting ambiguity in software requirement specification. In *2017 8th International Conference on Information Technology (ICIT)*. 209–213. (Cited on page 67.)
- SADIQ, M., HASSAN, T., AND NAZNEEN, S. 2017. Applying analytic hierarchy process in goal oriented requirements elicitation method for the prioritization of software requirements. In *2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)*. 1–5. (Cited on pages 7, 59, and 63.)
- SAILER, M., HENSE, J. U., MAYR, S. K., AND MANDL, H. 2017. How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in Human Behavior* 69, 371–380. (Cited on pages 85 and 86.)

- SAMER, R., ATAS, M., FELFERNIG, A., STETTINGER, M., FALKNER, A., AND SCHENNER, G. 2018. Group decision support for requirements management processes. In *Proceedings of the 20th Configuration Workshop, Graz, Austria, September 27-28, 2018*. 19–24. (Cited on pages 59, 65, 75, 81, 102, 119, and 166.)
- SAMER, R., FELFERNIG, A., AND STETTINGER, M. 2019. Towards issue recommendation for open source communities. In *2019 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019, Thessaloniki, Greece, October 14-17, 2019*. WI '19. ACM, 164–171. (Cited on pages 59, 65, 78, 79, 102, 143, 167, and 173.)
- SAMER, R., STETTINGER, M., ATAS, M., FELFERNIG, A., RUHE, G., AND DESHPANDE, G. 2019. New approaches to the identification of dependencies between requirements. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. 1265–1270. (Cited on pages 58, 59, 69, 85, 89, 102, 164, and 165.)
- SAMER, R., STETTINGER, M., AND FELFERNIG, A. 2020. Group recommender user interfaces for improving requirements prioritization. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*. UMAP '20. Association for Computing Machinery, New York, NY, USA, 221–229. (Cited on pages 52, 54, 59, 63, 64, 75, 101, 165, 170, 173, and 174.)
- SAMER, R., STETTINGER, M., AND FELFERNIG, A. 2021. An overview of recommendation technologies in software requirements engineering. Tech. rep., Graz University of Technology. (Cited on page 35.)
- SAMER, R., STETTINGER, M., FELFERNIG, A., FRANCH, X., AND A., F. 2020. Intelligent recommendation & decision technologies for community-driven requirements engineering. In *European Conference on Artificial Intelligence, Santiago de Compostela, Spain, August 29 - September 5, 2020*. ECAI '20. 3017–3025. (Cited on pages 37, 42, 72, 75, and 159.)
- SCHEIN, A. I., POPESCU, A., UNGAR, L. H., AND PENNOCK, D. M. 2002. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 253–260. (Cited on page 7.)
- SCHMID, K. 2000. Scoping Software Product Lines. In *Software Product Lines – Experience and Research Directions*. 513–532. (Cited on page 22.)
- SCHULZ-HARDT, S., BRODBECK, F., MOJZISCH, A., KERSCHREITER, R., AND FREY, D. 2006. Group decision making in hidden profile situations: Dissent as a facilitator of decision quality. *Journal of Personality and Social Psychology* 91, 6, 1080–1093. (Cited on pages 8, 12, 13, 63, 103, 106, 109, 110, 111, 165, 173, and 181.)
- SHAH, U. AND JINWALA, D. 2015. Resolving ambiguities in natural language software requirements: A comprehensive survey. *SIGSOFT Softw. Eng. Notes* 40, 5 (Sept.), 1–7. (Cited on page 102.)

- SHAO, F., PENG, R., LAI, H., AND WANG, B. 2017. DRank: A semi-automated requirements prioritization method based on preferences and dependencies. *The Journal of Systems and Software* 126, 141–156. (Cited on page 138.)
- SLANKAS, J. AND WILLIAMS, L. 2013. Automated extraction of non-functional requirements in available documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. 9–16. (Cited on pages 59 and 60.)
- SOMMERVILLE, I. 2010. *Software Engineering*, 9th ed. Addison-Wesley Publishing Company, USA. (Cited on page 1.)
- STANIK, C., HAERING, M., AND MAALEJ, W. 2019. Classifying multilingual user feedback using traditional machine learning and deep learning. *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), Jeju Island, South Korea*, 220–226. (Cited on pages 59, 61, 74, 163, and 175.)
- STANIK, C. AND MAALEJ, W. 2019. Requirements intelligence with OPENREQ analytics. In *2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju Island, South Korea*. 482–483. (Cited on pages 59, 61, 74, and 163.)
- STANIK, C., MONTGOMERY, L., MARTENS, D., FUCCI, D., AND MAALEJ, W. 2018. A simple NLP-based approach to support onboarding and retention in open source communities. *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 172–182. (Cited on pages 10, 59, 64, 65, 78, 102, 145, 148, and 167.)
- STETTINGER, M., FELFERNIG, A., LEITNER, G., AND REITERER, S. 2015. Counteracting anchoring effects in group decision making. In *User Modeling, Adaptation and Personalization*, F. Ricci, K. Bontcheva, O. Conlan, and S. Lawless, Eds. Springer, 118–130. (Cited on page 8.)
- STETTINGER, M., FELFERNIG, A., LEITNER, G., REITERER, S., AND JERAN, M. 2015. Counteracting serial position effects in the CHOICLA group decision support environment. In *20th ACM Conference on Intelligent User Interfaces (IUI2015)*. Atlanta, Georgia, USA, 148–157. (Cited on pages 86, 113, 132, 165, and 180.)
- STUMPTNER, M. 1997. An overview of knowledge-based configuration. *AICOM* 10, 2, 111–125. (Cited on page 21.)
- TARJAN, R. 1971. Depth-first search and linear graph algorithms. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*. SWAT '71. IEEE Computer Society, Washington, DC, USA, 114–121. (Cited on pages 55 and 108.)
- TEICHROEW, D. AND SAYANI, H. 1980. Computer-aided requirements engineering. In *Proceedings of the ACM 1980 Annual Conference*. ACM '80. Association for Computing Machinery, New York, NY, USA, 369–381. (Cited on page 3.)

-
- TIAN, Y., LO, D., XIA, X., AND SUN, C. 2015. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering* 20, 5 (Oct), 1354–1383. (Cited on pages 10, 133, and 144.)
- TINTAREV, N. AND MASTHOFF, J. 2007. A survey of explanations in recommender systems. In *2007 IEEE 23rd International Conference on Data Engineering Workshop*. 801–810. (Cited on pages 85 and 188.)
- TIWARI, S. AND LADDHA, M. 2017. Ucanalyzer: A tool to analyze use case textual descriptions. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 448–449. (Cited on pages 59 and 67.)
- TIZARD, J., WANG, H., YOHANNES, L., AND BLINCOE, K. 2019. Can a conversation paint a picture? Mining requirements in software forums. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 17–27. (Cited on pages 59 and 61.)
- TONELLA, P., SUSI, A., AND PALMA, F. 2013. Interactive requirements prioritization using a genetic algorithm. *Information and Software Technology* 55, 1, 173 – 187. (Cited on pages 59, 63, and 132.)
- TSANG, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press, London. (Cited on pages 26, 132, and 139.)
- TVERSKY, A. AND KAHNEMAN, D. 1975. *Judgment under Uncertainty: Heuristics and Biases*. Springer, Dordrecht, 141–162. (Cited on pages 63, 111, 173, and 180.)
- VAN METEREN, R. AND VAN SOMEREN, M. 2000. Using content-based filtering for recommendation. In *Proceedings of ECML 2000 Workshop: Matching Learning in Information Age*. 47–56. (Cited on pages 4 and 43.)
- VOGELSANG, A. AND FUHRMANN, S. 2013. Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study. *2013 21st IEEE International Requirements Engineering Conference, RE 2013 - Proceedings*, 267–272. (Cited on pages 6 and 68.)
- VON WINTERFELDT, D. AND EDWARDS, W. 1986. *Decision Analysis and Behavioral Research*. Cambridge University Press. (Cited on page 81.)
- WALLENIUS, J., DYER, J., FISHBURN, P., STEUER, R., ZIONTS, S., AND DEB, K. 2008. Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead. *Manage. Sci.* 54, 7 (July), 1336–1349. (Cited on page 64.)
- WANG, S. AND MANNING, C. D. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*. ACL '12. Association for Computational Linguistics, Stroudsburg, PA, USA, 90–94. (Cited on page 153.)
- WANG, Y., CHAN, S. C.-F., AND NGAI, G. 2012. Applicability of demographic recommender system to tourist attractions: A case study on trip advisor. In *Proceedings of the The 2012*

- IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 03*. WI-IAT '12. IEEE Computer Society, Washington, DC, USA, 97–101. (Cited on page 43.)
- WIEGERS, K. 2003. *Software Requirements*. Microsoft Press. (Cited on page 132.)
- WILLIAMS, G. AND MAHMOUD, A. 2017. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 1–10. (Cited on pages 59 and 61.)
- WILMINK, M. AND BOCKISCH, C. 2017. On the ability of lightweight checks to detect ambiguity in requirements documentation. In *Requirements Engineering: Foundation for Software Quality*. Springer, 327–343. (Cited on pages 59 and 67.)
- WINKLER, J. AND VOGELSANG, A. 2016. Automatic classification of requirements based on convolutional neural networks. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 39–45. (Cited on pages 6, 59, 60, and 127.)
- XIE, M., LAKSHMANAN, L., AND WOOD, P. 2010. Breaking out of the Box of Recommendations: From Items to Packages. In *4th ACM Conference on Recommender Systems*. Barcelona, Spain, 151–158. (Cited on page 20.)
- XU, D. AND TIAN, Y. 2015. A comprehensive survey of clustering algorithms. *Annals of Data Science* 2, 2, 165–193. (Cited on page 57.)
- XU, J., YAO, Y., TONG, H., TAO, X., AND LU, J. 2015. Ice-breaking: Mitigating cold-start recommendation problem by rating comparison. In *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. AAAI Press, 3981–3987. (Cited on page 7.)
- XUAN, J., JIANG, H., REN, Z., AND LUO, Z. 2012. Solving the large scale next release problem with a backbone-based multilevel algorithm. *IEEE Transactions on Software Engineering* 38, 5, 1195–1212. (Cited on page 133.)
- XUAN, J., JIANG, H., REN, Z., AND ZOU, W. 2012. Developer prioritization in bug repositories. In *Proceedings of the 34th International Conference on Software Engineering*. ICSE '12. IEEE Press, Piscataway, NJ, USA, 25–35. (Cited on pages 102, 132, 144, and 165.)
- YANG, H.-L. AND WANG, C.-S. 2009. Recommender system for software project planning one application of revised cbr algorithm. *Expert Syst. Appl.* 36, 5 (July), 8938–8945. (Cited on pages 59 and 70.)
- ZHANG, B. AND ZHOU, H. 2017. Brief announcement: Statement voting and liquid democracy. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC '17. Association for Computing Machinery, New York, NY, USA, 359–361. (Cited on pages 54, 64, and 186.)
- ZHANG, Y., HARMAN, M., AND MANSOURI, S. A. 2007. The multi-objective next release problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. GECCO

- '07. Association for Computing Machinery, New York, NY, USA, 1129–1137. (Cited on pages 59, 70, and 132.)
- ZHAO, X., XIA, L., ZHANG, L., DING, Z., YIN, D., AND TANG, J. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys '18. Association for Computing Machinery, New York, NY, USA, 95–103. (Cited on page 186.)
- ZHENG, G., ZHANG, F., ZHENG, Z., XIANG, Y., YUAN, N. J., XIE, X., AND LI, Z. 2018. Dnn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*. WWW '18. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 167–176. (Cited on page 186.)
- ZHOU, Z. 2017. A brief introduction to weakly supervised learning. *National Science Review* 5, 1 (08), 44–53. (Cited on page 69.)