New Approaches to the Identification of Dependencies between Requirements

Ralph Samer
Graz University of Technology
Graz, Austria
rsamer@ist.tugraz.at

Martin Stettinger
Graz University of Technology
Graz, Austria
martin.stettinger@ist.tugraz.at

Muesluem Atas
Graz University of Technology
Graz, Austria
muatas@ist.tugraz.at

Alexander Felfernig
Graz University of Technology
Graz, Austria
alexander.felfernig@ist.tugraz.at

Guenther Ruhe
University of Calgary
Calgary, Canada
ruhe@ucalgary.ca

Gouri Deshpande
University of Calgary
Calgary, Canada
gouri.deshpande@ucalgary.ca

***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 paper, 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*, *Naive 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 data set. The evaluation shows that the recommendation approach based on *Random Forest* using probabilistic features achieves the best prediction quality of all approaches (F1: 0.89).**

***Keywords*-Requirements Engineering, Content-based Recommender Systems, Machine Learning**

I. Introduction

Recommender Systems (RS) are decision support systems which help users to select a well-collected set of items matching their needs and preferences [1], [15]. 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 [9], [12]. RS can support stakeholders in different RE tasks such as, *requirements definition/elicitation, release decisions, stakeholder identification*, and *dependency detection* [12], [14].

Usually, a software project consists of hundreds of different requirements which are often related to each other. Single requirements elicitation methods such as interviews [5] 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 [11], [12]. 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 [16]. 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 [10]. 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 [3], [7]. The work of Chitchyan et al. describes an NLP-based approach which assists in the identification of dependencies between requirements on a semantic level [4]. Ninaus et al. [14] present an RE tool which applies recommendation techniques to support stakeholders in RE tasks. Their tool also includes a basic dependency RS which recommends similar requirements that are treated as potential dependencies. Atas et al. [2] 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 paper are the following. The work presented in this paper extends the basic approach of Atas et al. [2] 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. [2] and Ninaus et al. [14], 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 paper is structured as follows. In Section II, 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 III presents the used pre-processing and feature extraction techniques. In Section IV, 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 V. Finally, we conclude the paper and provide a brief outlook towards future work in Section VI.

II. User Study & Dataset

We used a dataset[1] 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 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 [13], 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 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.

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

Table I 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.

Table I
Dependencies found by experts and students.

		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**	-

III. 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.

A. 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 **v** for a requirement pair (r_x, r_y). Formula 1 provides a formal representation of the feature vector **v** whereby each TF-IDF value of a 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 "∪".

$$\mathbf{v}(r_x, r_y) = \bigcup_{i \in ngrams(r_x)} \{TFIDF(i)\} \cup \bigcup_{j \in ngrams(r_y)} \{TFIDF(j)\} \tag{1}$$

B. Extraction of Probabilistic Features

As an alternative feature representation, we also used features which take aspects from the area of *Information Theory* into account [8]. In the remainder of this paper, 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 2).

$$pv(T_x, T_y) = \sum_{w \in T_x} \sum_{v \in T_y} log \frac{p(w, v)}{p(w) * p(v)} \tag{2}$$

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 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.

IV. 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].

A. 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, Naive Bayes, Random Forest,* and *k-Nearest Neighbor* (k-NN) and evaluated their performance (see Section V). 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

[2]Scikit-learn: http://scikit-learn.org

and r_y. The training pairs/samples were then used to learn a prediction model. To generate all training pairs, 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 III-A). In the case of *probabilistic features*, we used the five computed features described in Section III-B. 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.

B. 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 [6]. 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 which 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 3).

$$\mathbf{d}'_x = \Sigma_k^{-1} U_k^T \mathbf{d}_x \tag{3}$$

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.

V. Evaluation & Discussion

To evaluate both approaches described in Section IV, 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 III). 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 0.8[3] 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 II 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 reasonably by all algorithms. In particular, high scores in terms of precision and F1 can be observed for *Linear SVM* (Precision: 0.997, F1: 0.695) and *Naive Bayes* (0.901, F1: 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 V) 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 naivety and can not develop a good sense in order to distinguish between those requirements that are just similar versus those requirements that are really dependent.

Since LSA requires a document-term matrix as input, it cannot be combined with our probabilistic features. *Naive Bayes* is also supposed to be used only with TF-IDF features (or term frequencies). Hence, we only evaluated the RS based on the other three classifiers. 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: 0.897) and could even significantly outperform the baseline approach in terms of recall. This behaviour 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.

Table II
SCORES OF THE DIFFERENT ALGORITHMS (PRECISION [P], RECALL [R], AND F1 SCORE [F1]). THE HIGHEST SCORES ARE HIGHLIGHTED.

	TF-IDF Features			**Probabilistic Features**		
Algorithm	P	R	F1	P	R	F1
LSA (Baseline)	0.600	**0.818**	0.692	–	–	–
Naive 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**

VI. CONCLUSION & FUTURE WORK

Conclusion. In this paper, 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 [10]. The first RS was based on classification and the second was based on *Latent Semantic Analysis* (LSA). The used classifiers (*Naive 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 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

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

even significantly outperform LSA's high recall benchmark of 0.818. Consequently, the main finding of this work 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 a dependency $r_x \rightarrow r_y$ 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.

Acknowledgment

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OpenReq (732463).

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. 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):734–749, June 2005.

[2] Muesluem Atas, Ralph Samer, and Alexander Felfernig. Automated identification of type-specific dependencies between requirements. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 688–695, Dec. 2018.

[3] Pär Carlshamre, Kristian Sandahl, Mikael Lindvall, Björn Regnell, and Johan Nattoch Dag. An industrial survey of requirements interdependencies in software product release plannin. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 84–, Washington, DC, USA, 2001. IEEE Computer Society.

[4] Ruzanna Chitchyan and Awais Rashid. Tracing requirements interdependency semantics. In *Workshop on Early Aspects*, Jan 2006.

[5] Alan Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 176–185, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Scott Deerwester, Susan T. Dumais, George Furnas, Thomas Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 09 1990.

[7] Gouri Deshpande. Sreyantra: Automated software requirement inter-dependencies elicitation, analysis and learning. In *41st International Conference on Software Engineering (ICSE'19)*, Canada, 2019.

[8] Benjamin Van Durme and Ashwin Lall. Streaming pointwise mutual information. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 1892–1900, USA, 2009. Curran Assoc. Inc.

[9] Alexander Felfernig, Gerald Ninaus, Harald Grabner, Florian Reinfrank, Leopold Weninger, Denis Pagano, and Walid Maalej. An overview of recommender systems in requirements engineering. In *Managing Requirements Knowledge*, chapter 14, pages 315–332. Springer, 2013.

[10] Stefan Ferber, Jürgen Haag, and Juha Savolainen. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In Gary J. Chastek, editor, *Software Product Lines*, pages 235–256, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[11] Dean Leffingwell. Calculating your return on investment from more effective requirements management. *American Programmer*, 10(4):13–16, 1997.

[12] Bamshad Mobasher and Jane Cleland-Huang. Recommender systems in requirements engineering. *AI Magazine*, 32(3):81–89, Jun. 2011.

[13] Jamie Murphy, Charles F. Hofacker, and Richard Mizerski. Primacy and recency effects on clicking behavior. *Journal of Computer-Mediated Communication*, 11(2):522–535, 2006.

[14] Gerald Ninaus, Florian Reinfrank, Martin Stettinger, and Alexander Felfernig. Content-based recommendation techniques for requirements engineering. In *2014 IEEE 1st International Workshop on AI for Requirements Engineering (AIRE)*, pages 27–34, Aug 2014.

[15] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.

[16] Günther Ruhe. *Product release planning: methods, tools and applications*. CRC Press, 2010.