Towards Feature-Aware Retrieval of Refinement Traces

Patrick Rempel, Patrick Mäder, and Tobias Kuschke
Ilmenau Technical University
Department of Software Systems
Ilmenau, Gernany
{patrick.rempel|patrick.maeder|tobias.kuschke}@tu-ilmenau.de

***Abstract*—Requirements traceability supports practitioners in reaching higher project maturity and better product quality. To gain this support, traces between various artifacts of the software development process are required. Depending on the number of existing artifacts, establishing traces can be a time-consuming and error-prone task. Additionally, the manual creation of traces frequently interrupts the software development process. In order to overcome those problems, practitioners are asking for techniques that support the creation of traces (see Grand Challenge: Ubiquitous (GC-U)). In this paper, we propose the usage of a graph clustering algorithm to support the retrieval of refinement traces. Refinement traces are traces that exist between artifacts created in different phases of a development project, e.g., between features and use cases. We assessed the effectiveness of our approach in several TraceLab experiments. These experiments employ three standard datasets containing differing types of refinement traces. Results show that graph clustering can improve the retrieval of refinement traces and is a step towards the overall goal of ubiquitous traceability.**

***Index Terms*—requirements traceability; graph clustering; trace retrieval; TraceLab experiment; ubiquitous challenge (GC-U);**

I. Introduction

Requirements traceability supports practitioners in reaching higher project maturity and better product quality. To gain this support, traces between various artifacts of the software development process are required and must be established. Depending on the number of existing artifacts, establishing traces can be a time-consuming and error-prone task. Additionally, the manual creation of traces frequently interrupts the software development process. In order to overcome those problems, practitioners are asking for techniques that support the creation of traces [1], [2], [3]. The community of researchers working on requirements traceability problems recently defined a catalog of common and important research goals, the so-called grand challenges of requirements traceability [4], [5]. The work presented here refers to the ubiquitous traceability challenge (GC-U). The challenge is described as “establishing traceability manually is open to human error and inconsistency, and its quality is only as good as the efforts of its weakest human link” [4], [5]. As a consequence, “traceability should not be the goal of software and systems development, it should not force a break in the engineering”[1].

To address this challenge, researchers developed semi-automated techniques that provide support for trace creation. These techniques are based on information retrieval methods such as the Vector Space Model (VSM), Latent Semantic Indexing (LSI), probabilistic networks [6], [7], [8], [9], [10], [11], and feature location techniques [12]. Despite considerable improvements, automated trace retrieval techniques still only return 85–90% of the targeted traces at low precision rates of usually 15–25%.

In this paper, we are focusing on the retrieval of refinement traces. Refinement traces are traces that exist between artifacts created in different phases of a development project, e.g., between features and use cases. Our hypothesis was that information about the cohesion of artifacts within one refinement level could help in improving the retrieval process of traces among refinement levels and ultimately lead to better trace retrieval tools. We propose the usage of a graph clustering algorithm that groups artifacts within a refinement level into different clusters by computing cohesion within the clusters and coupling among clusters.

Our research question was: Can knowledge about the clustering of artifacts that belong to the same refinement level of a development project help in discovering traces between those artifacts and artifacts belonging to another refinement level?

The goal of our work was to assess the effect of clustering for the trace retrieval process. Accordingly, we combined the clustering with the widely used Vector Space Model (VSM) [13], [6]. We assessed the effectiveness of our approach in several TraceLab experiments. The results of our VSM + clustering approach were compared on three different datasets with the performance of the stand-alone VSM and LSI algorithms. We chose VSM and LSI as reference approaches for two reasons. First, VSM and LSI are industry standard information retrieval techniques. Second, LSI clusters the term-document matrix into latent semantic spaces and therefore has similarities with our approach. As extensively discussed in [14], LSI with singular value decomposition and clustering with Fiedler clustering [15] show a high degree of structural similarity. Although, LSI can only leverage term-document connections, while Fiedler clustering allow the inclusion of document-document similarities.

[1] http://www.coest.org/index.php/research-directions/grand-traceability-challenges, last accessed February 2013

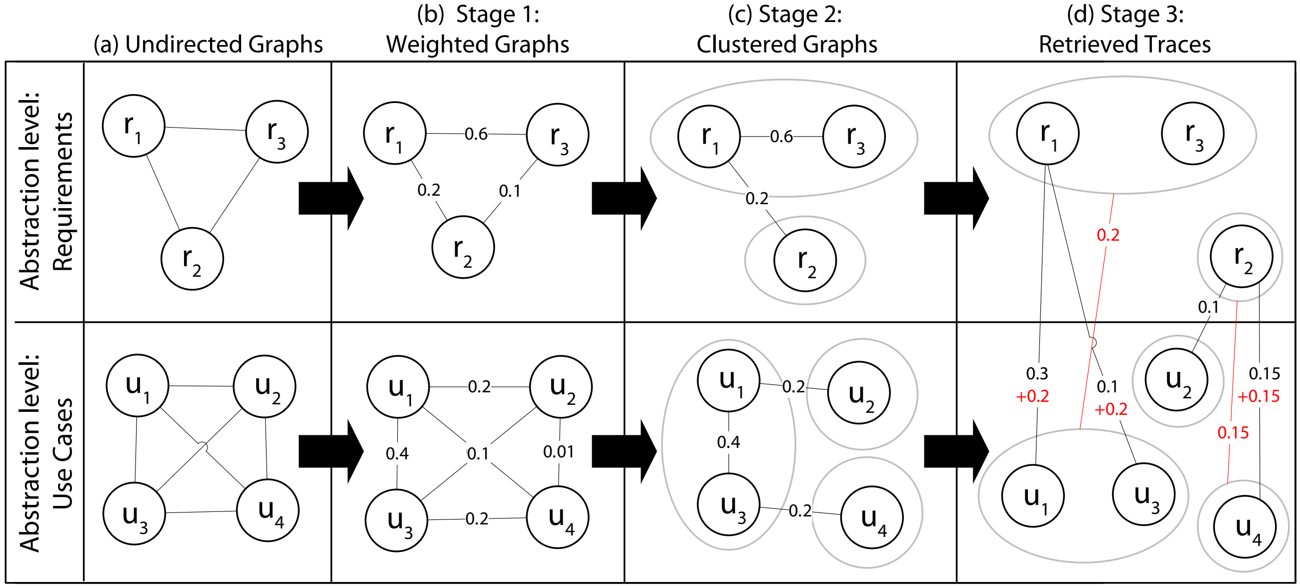

Fig. 1. Overview of the approach that uses graph clustering per artifact abstraction level to improve trace retrieval across abstraction levels.

II. Approach

In this section we discuss the general ideas and assumptions, which led to the hypothesis that graph clustering methods can have a positive impact on existing methods for automated trace retrieval of refinement traces. Furthermore, we propose an approach that incorporates graph clustering in the automated retrieval of refinement traces.

A. Context and General Idea

Typically, complex software products are developed in many iterations (software releases) as the complete set of all software product requirements is not entirely known from the beginning and continuously changes over time. Every iteration consists of different development phases such as analysis, design, development, and test. Each phase produces different artifact types such as requirements, use cases, source code, test cases. Although, software development is fragmented into various releases, development phases, and artifact types, the software product itself consists of features, which were implemented to serve the needs of stakeholders. Every feature is developed in one or many software releases through all software development phases. Thus, a single software development phase produces a set of artifacts, which represents the feature on a certain abstraction level (e.g., a set of elements in a UML class diagram represents a feature on the design level, a set of source code classes represents a feature on the implementation level). Thus, every feature of a software product can exists through all artifact abstraction level of a development project. These abstraction levels correspond to software development phases. Artifacts of the same feature at different abstraction levels should be connected by refinement traces in order to facilitate their consistent maintenance over the life of the developed product. We therefore hypothesize that the identification of features at different abstraction levels provides additional information, which can be leveraged for the automated retrieval of refinement traces.

B. Stage 1: Weighted Artifact Graphs

Initially, all n artifacts that belong to the same software development phase are represented as an undirected fully connected graph $G = (V, E)$ with $V = \{a_i | a_i$ is an individual artifact$\}$ and $E = \{e_i | e_i$ is a dependency relation between a_i and $a_j, 1 \leq i, j \leq n\}$ as proposed by Chen et al. [16]. Figure 1a shows two such undirected simplex graphs for the two abstraction levels requirements and use cases.

We define a weight $w(e_{i,j})$ for each edge of the simplex graph. The weight expresses how strong two artifacts of the same abstraction level are connected. For the evaluation of the proposed clustering approach, we utilize the Vector Space Model (VSM) with the *tf-idf* weighting scheme [13], where *tf* is the term frequency and *idf* is the inverse document frequency. The *tf-idf* weight is computed as:

$$w_{i,j} = tf_{ij} \times \log\left(\frac{n}{n_i}\right) \qquad (1)$$

where tf_{ij} is the term frequency per artifact, n_i is the number of artifacts in which the term i occurs, and n is the total number of artifacts.

To calculate the similarity of all possible artifact relations per artifact abstraction level, a cosine similarity score is calculated for every possible pair of artifacts:

$$sim(a_j, a_k) = \frac{\sum_{i=1}^{m} w_{ij} \times w_{ik}}{\sqrt{\sum_{i=1}^{m} w_{ij}^2} \times \sqrt{\sum_{i=1}^{m} w_{ik}^2}} \qquad (2)$$

where a_j and a_k are artifacts. Figure 1b illustrates the two resulting weighted graphs.

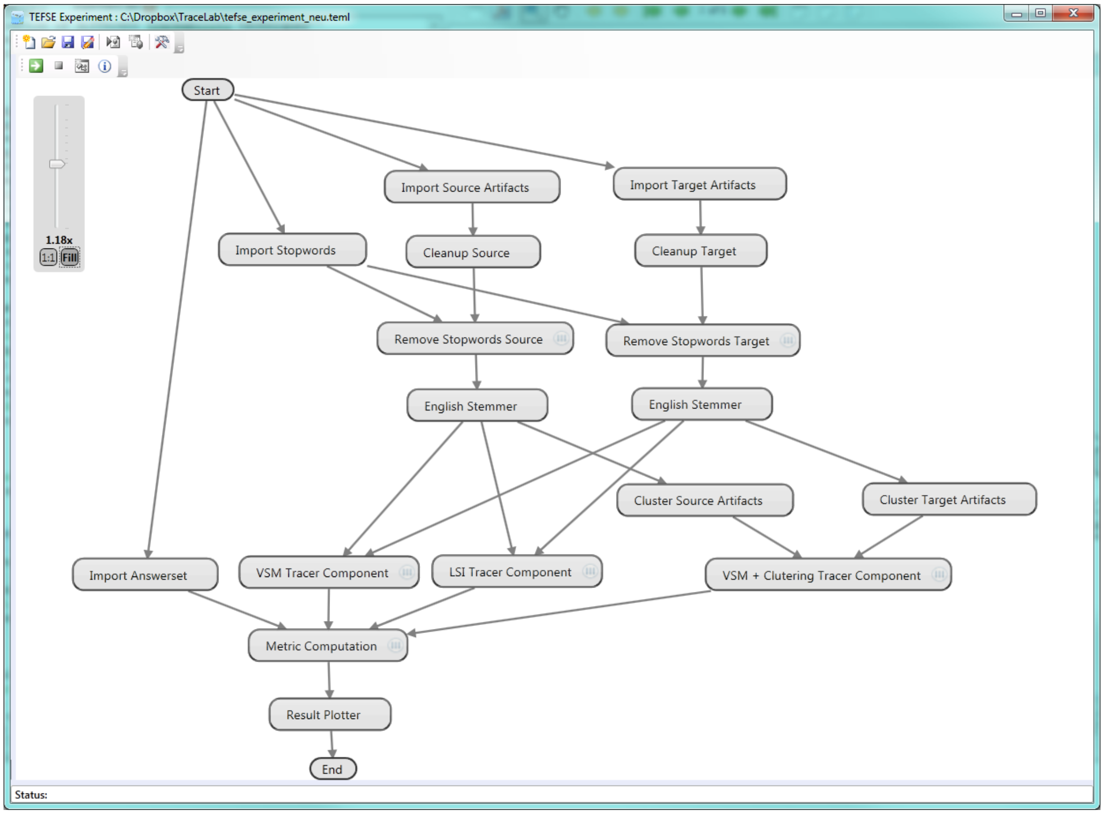

Fig. 2. TraceLab workflow for assessing the performance of our VSM + Clustering approach.

C. Stage 2: Clustered Artifact Graphs

The resulting weighted graph (see Figure 1b) represents artifacts and their dependencies within the same artifact abstraction level (e.g., high-level requirements). To identify artifacts that belong to the same feature we apply a spectral partitioning method [17], which groups the artifact graph of each abstraction level into clusters. Therefore, each weighted artifact graph is transformed into an adjacency matrix A, where the previously calculated artifact similarity weights are used as adjacency scores. The corresponding Laplacian matrix L is defined as:

$$L = D - A \tag{3}$$

where D is a diagonal matrix containing the row sums of A. Fiedler [15] proved that the eigenvector of the second smallest eigenvalue of the Laplacian matrix, which is often referred to as Fiedler eigenvector, is suitable to divide a graph into a sub-graph with maximally inter-connected vertices and a sub-graph with minimally inter-connected vertices. The rows of the Fiedler eigenvector with the same sign (e.g., positive) are put into the same cluster. For each sub-graph, the described Fiedler method can be repeated to further refine artifact clusters. The result of this stage of the approach is illustrated in Figure 1c.

D. Stage 3: Retrieved Refinement Traces

Once, artifact clusters per abstraction level have been identified (see Figure 1c), the actual trace retrieval process is performed in three steps. The goal is to retrieve refinement traces between artifacts from two different abstraction levels.

Step 1 The similarity for all possible pairs of high-level vs. low-level artifacts is computed.

Step 2 The similarity between all pairs of high-level vs. low-level clusters of artifacts is computed.

Step 3 The results of the first two steps are combined. A cluster bonus is added to artifact pairs (Step 1) that belong to that high-level – low-level cluster pair with the highest cluster similarity score per high-level cluster (Step 2). After the cluster score is added, the resulting artifact pair similarity weights are used to retrieve artifact traces.

The result of this final stage of the approach is illustrated in Figure 1d.

III. Case Study

We performed three experiments to evaluate the effectiveness of our clustering approach in comparison to the VSM and the LSI approach. Our experiments were implemented and executed with TraceLab. "TraceLab is an experimental workbench for designing, constructing, and executing traceability experiments, and facilitating the rigorous evaluation of different traceability techniques" [18]. We chose TraceLab because it offered an optimal environment for rapidly evaluating and evolving our hypotheses.

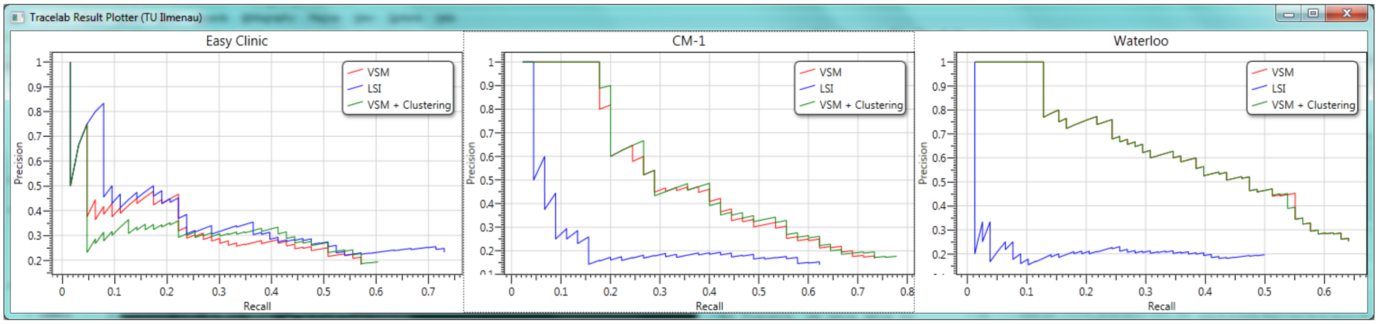

Fig. 3. Precision-recall graphs for the three evaluated datasets. Red lines show the quality of retrieved traces based on VSM. Blue lines show the quality of retrieved traces based on LSI. Green lines show the quality of retrieved traces with support of the proposed VSM + Clustering approach.

A. *Datasets*

We used three different datasets for conducting our experiments: EasyClinic, CM-1, and the Waterloo project. These datasets are commonly applied for traceability and especially trace retrieval research. The first two are available from COEST[2]. The Waterloo dataset was part of the Tracelab source code[3]. The datasets contain different artifact types, allowing us to evaluate our approach in a broader context. For CM-1 we retrieved traces between abstract requirements and use cases. For EasyClinic, we retrieved traces between use cases and functional test cases. Finally, for the Waterloo project the goal was to retrieve traces between features and use cases.

B. *Experimental Design*

To perform the experiment we modeled the experiment workflow depicted in Figure 2. We used existing TraceLab importer components to load source artifacts, target artifacts, and the answer matrix with all correct traces. Additionally, we used existing preprocessing components of the TraceLab package to pre-process artifacts, remove stop words, and to stem words. We developed our own clustering component, which created artifact clusters based on the clustering algorithm described in Section II. Additionally, we developed two tracer components, which we used to compare simple term similarity analysis with our clustering approach.

IV. Results

In order to assess and compare the performance of our approach for different datasets, we use the following standard metrics [19]:

- *Recall R:* measures the fraction of relevant documents that are correctly retrieved;
- *Precision P:* measures the fraction of retrieved documents that are relevant; and
- *Average Precision AP:* measures how well a traceability technique retrieves relevant documents at the top of the ranked retreval results.

Figure 3 visualizes the results of our experiment as precision-recall graphs (see Figure 3). Table I shows the *Average Precision* of the three approaches (VSM, LSI, and VSM + Clustering) for all datasets. We applied the *aggregation method*[4] to summarize the trace retrieval results across all queries. Table I shows that the clustering approach provides similar results to the VSM approach for all datasets in terms of average precision. While LSI outperforms VSM and VSM + Clustering for the EasyClinic dataset. VSM and VSM + Clustering outperform LSI for the CM-1 and the Waterloo datasets. Figure 3 shows that the results of the three different approaches (VSM, LSI, and VSM + Clustering) also vary across the three datasets in terms of recall and precision. While LSI appears to be most appropriate for EasyClinic, its results for CM-1 and Waterloo are less promising. VSM and VSM + Clustering outperform LSI for CM-1 and Waterloo.

TABLE I
Average precision AP (computed with aggregation method) of traces retrieved with VSM, LSI, and VSM + Clustering for the three datasets.

Dataset	**VSM**	**LSI**	**VSM + Clustering**
EasyClinic	0.264	0.304	0.257
CM-1	0.425	0.196	0.434
Waterloo	0.472	0.182	0.472

For the EasyClinic dataset, the average precision of VSM + Clustering is lower than that of LSI. The lower average precision of the Clustering approach for the EasyClinic dataset is the result of higher precision rates at very low recall levels. That is negligible as primarily the precision at high recall rates is practically relevant for the automated retrieval of traces. At high recall levels, the clustering approach produces similar precision rates to LSI and outperforms VSM. For the CM-1 dataset, the average precision of VSM and VSM +

[2] http://www.coest.org, last accessed February 2013

[3] This dataset contains several specifications of a VoIP system produced by 3- or 4-person teams as term project for the requirements engineering course (CS445: Software Requirements Specification) in the Software Engineering program of the School of Computer Science at the University of Waterloo, Ontario, Canada.

[4] http://www.coest.org/index.php/research-directions/evaluation-methods, last accessed February 2013

Clustering outperform LSI. Furthermore, VSM + Clustering slightly outperforms VSM. Especially at high recall level, the VSM + Clustering approach provides higher precision than the stand-alone VSM. For the Waterloo dataset, the average precision of VSM and VSM + Clustering are equal. In fact, the clustering algorithm identified only a single cluster and thus made the effect of clustering negligible.

The results of these experiments partly confirmed our hypothesis on recoverable feature clusters and their positive impact on the trace retrieval. However, more experiments are required to draw a general conclusion on the appropriateness of the proposed approach.

V. Experiences with TraceLab

Similar to a rapid prototyping environment in other engineering disciplines, the use of TraceLab enabled us to develop a research idea on the white board and to test it within hours. We added our ideas as new components to TraceLab. Due to existing reusable components such as Importers, Preprocessors, Exporters, Visualizers, and Tracers, we were able to quickly setup a complex experiment.

VI. Discussion and Future Work

In this paper, we proposed the usage of a graph clustering algorithm to support the retrieval of refinement traces, existing between artifacts created in different phases of a development project. We assessed the effectiveness of our approach in several TraceLab experiments. These experiments employ three standard datasets containing differing types of refinement traces. Results show that graph clustering can improve the retrieval of refinement traces and can make the overall goal of ubiquitous traceability a bit more realistic. The comparison of the approaches for the different datasets also shows that there is no best trace retrieval method. While LSI produced the best results for the EasyClinic dataset, VSM and VSM + Clustering (the proposed method) outperformed LSI for the CM-1 and the Waterloo datasets. Furthermore, VSM + Clustering produced equal precision values to LSI at high recall values. The results of our experiments suggest that the proposed clustering approach provides reasonable precision rates regardless of whether the datasets is clustered or not, while LSI seems more suited for clustered datasets and the stand-alone VSM more suited for unclustered datasets.

Future work will focus on extending our experiments to more thoroughly assess the proposed approach. Furthermore, it could be beneficial to consider additional information about artifacts, like hierarchy in a document.

Acknowledgment

The authors would like to thank the TraceLab team for their great work. We are funded by the German Research Foundation (DFG): Ph49/8-1 and the German Ministry of Education and Research (BMBF): grant 16V0116.

References

[1] J. Cleland-Huang, R. Settimi, E. Romanova, B. Berenbach, and S. Clark, “Best practices for automated traceability,” *Computer*, vol. 40, no. 6, pp. 27–35, 2007.

[2] P. Mäder, O. Gotel, and I. Philippow, “Motivation matters in the traceability trenches,” in *Proceedings 17th IEEE International Requirements Engineering Conference (RE09)*. IEEE Computer Society, 2009, pp. 143–148. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/RE.2009.23

[3] E. Bouillon, P. Mäder, and I. Philippow, “A survey on usage scenarios for requirements traceability in practice,” in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, J. Doerr and A. L. Opdahl, Eds. Springer Berlin Heidelberg, 2013, vol. 7830.

[4] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grnbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, *The Grand Challenge of Traceability (v1.0)*. Springer-Verlag, 2012, pp. 343–412.

[5] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol, “The quest for ubiquity: A roadmap for software and systems traceability research,” in *20th IEEE International Conference on Requirements Engineering*, M. P. E. Heimdahl and P. Sawyer, Eds. IEEE, 2012, pp. 71–80.

[6] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, “Recovering traceability links between code and documentation,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[7] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, “Enhancing an artefact management system with traceability recovery features,” in *20th IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 306–315.

[8] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, “Advancing candidate link generation for requirements tracing: The study of methods,” *IEEE Trans. Softw. Eng.*, vol. 32, no. 1, pp. 4–19, 2006.

[9] A. Marcus, J. I. Maletic, and A. Sergeyev, “Recovery of traceability links between software documentation and source code,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 5, pp. 811–836, 2005.

[10] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, “Goal-centric traceability for managing non-functional requirements,” in *27th International Conference on Software Engineering*. IEEE, 2005, pp. 362–371.

[11] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou, “Utilizing supporting evidence to improve dynamic requirements traceability,” in *13th IEEE International Conference on Requirements Engineering*. IEEE, 2005, pp. 135–144.

[12] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. DeLucia, “On integrating orthogonal information retrieval methods to improve traceability link recovery,” in *Intn’l Conf. on Software Maintenance (ICSM’11)*, 2011, pp. 133–142.

[13] G. Salton, A. Wong, and C.-S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[14] B. Hendrickson, “Latent semantic analysis and fiedler retrieval,” *Linear Algebra and its Applications*, vol. 421, no. 2, pp. 345–355, 2007.

[15] M. Fiedler, “A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory,” *Czechoslovak Mathematical Journal*, vol. 25, no. 4, pp. 619–633, 1975.

[16] K. Chen, W. Zhang, H. Zhao, and H. Mei, “An approach to constructing feature models based on requirements clustering,” in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. IEEE, 2005, pp. 31–40.

[17] A. Pothen, H. D. Simon, and K.-P. Liou, “Partitioning sparse matrices with eigenvectors of graphs,” *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 3, pp. 430–452, 1990.

[18] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. Huffman Hayes *et al.*, “Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions,” in *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012, pp. 1375–1378.

[19] G. Salton, *Automatic Text Processing: The Transformation, Analysis,*