# Clustering Source Code Elements by Semantic Similarity Using Wikipedia

Mirco Schindler, Oliver Fox, Andreas Rausch
Department of Informatics - Software Systems Engineering
Clausthal University of Technology
38678 Clausthal-Zellerfeld, Germany
Email: {mirco.schindler, oliver.fox, andreas.rausch}@tu-clausthal.de

***Abstract*—For humans it is no problem to determine if two words have a high or low semantic similarity in a given context. But is it possible to support a software developer or architect by using semantic data extracted from source code in the same way other relations like typical source code relations do?**

**To answer this question we developed an approach to compute the semantic similarity by using Wikipedia as a textual corpus. In a case study we demonstrate this approach with a manageable software system. The results of using semantic similarities are compared with the outcome of using source code relations instead.**

## I. Introduction

Software engineering techniques like Domain Driven Design [1] or Model Driven Design [2] use natural language terms for naming source code elements. In general all these terms have a specific semantic meaning and can be linked intuitively by human beings.

In this paper we investigate if it is possible to use the semantic similarity between two words to support developers or architects that are designing or maintaining software systems, like the use of typical source code relations in the field of modularization. We had the goal to develop an approach in which the measurement of the semantic similarity is computed straightforward. We explicitly wanted to avoid the generation of an extra textual corpus or the usage of natural language processing techniques. Because it is very time-consuming and an enormous amount of input data is needed to generate an own textual corpus. In case of natural language processing often procedures like a stemming service for example are required. With source code elements that include terms in different languages, it is not that easy to get such a service for each language. Because of that we wanted to find a way to compute a semantic similarity which is based on natural language documents and could be used without any critical preprocessing.

The rest of the paper is structured as follows. In Section II other approaches using semantic analysis techniques are presented, followed by Section III, which presents our approach for clustering source code elements by semantic similarities using Wikipedia. A part of case studies is presented in Section IV which was executed using an example system. Its results were compared to a clustering, using classical source code relations. In Section V the findings are discussed and a conclusion and an outlook for future work and use cases to support software developer or architects is given.

## II. Related Work

In 1997 Anquetil and Lethbridge [3] wanted to retrieve informations about the decomposition of a software system by analyzing file names.

A related approach is the work of Kuhn, Ducasse and Gîrba [4]. They used the source code to extract semantic information from naming and comments and create a Latent Semantic Indexing (LSI) for a given source code scope that gets clustered afterwards. They performed a series of different clustering algorithms and decided to use a hierarchical average-linkage clustering.

LSI is a very popular information retrieval technique that locates linguistic topics in a set of documents. In [5] LSI is also used to rank concepts that were manually extracted from the source code. These concepts and their underlying source code elements are then processed via formal concept analysis (FCA) to create a lattice with selected attributes for them.

Similar to our clustering process Santos et. al. [6] introduce an approach containing two steps. In the first step they generate a term-document representation of a source code element and define a similarity measure based on this representation. In a second step these elements are clustered. In [6] LSI is used for creating the source code elements' representation and in contrast to our approach they apply a simple agglomerative hierarchical clustering algorithm instead of our top down approach.

A similar approach is utilized by Maletic and Marcus [7]. They use Latent Semantic Analysis (LSA) to extract semantic information about the usage of words in source code.

Other information retrieval techniques are used by Bavota et. al. in [8]. They wanted to improve software modularization by analyzing latent topics and associations in source code. To capture coupling between source code elements they use Relational Topic Models (RTM) in this article and grouped those elements by a technique related to a k-nearest neighbor approach. In [9] they used coupling metrics like Information-Flow based Coupling (ICP) and Conceptual Coupling Between Calsses (CCBC), which contains both structural and semantic coupling associations. For grouping they use the extraction of

978-1-4673-7064-6/15 $31.00 © 2015 IEEE
DOI 10.1109/RAISE.2015.10

class chains and define a coupling metric between these class chains to optimize the modularization.

Another significant technique to extract semantic information especially, for domain driven design in software engineering, is ontology extraction. One approach which uses this technique was presented by Kof [10]. He analyzed the documents of a software project with linguistic methods that utilize the sentence-structure to generate an ontology of the domain model.

All these approaches have in common, that they use the words of the application's domain model and no external textual corpus. A textual corpus is a large set of writings in one language that can be searched through for linguistic research purposes. The underlying data-structure is optimized for a special linguistical corpus-method in the most cases. Many universities and other linguistic facilities maintain their own corpora created on the base of books, scientific papers, encyclopedias and magazines etc.. In [11] the "WikiRelate!" project is described by Strube and Ponzetto, who do some research on using Wikipedia as corpus. In their paper they describe a set of classical corpus methods and new methods, that just work on the special structure of Wikipedia. In [12] an approach is presented that uses the Wikipedia corpus to find semantic relations between two given tags in the field of Web 2.0.

## III. Approach

The initial idea for the approach is based in the Bachelor Thesis of Oliver Fox [13]. After evaluating his approach by performing several case studies it was adapted to the version presented in this paper.

The approach consists of two steps. In the first step the semantic similarity between all elements using a textual corpus is computed. In a second step these elements are clustered to group those with a high similarity together. In this approach we use a graph representation of a software system, in which each class or interface is mapped to a node of the graph. The edge weight represents the similarity of the corresponding nodes.

In general corpus techniques are one of the established methods in the field of semantic analysis in linguistics. To calculate the semantic similarity Wikipedia is used as textual corpus. As mentioned in Section II, Wikipedia offers some advantages like multilingualism, number of articles or their actuality towards other textual corpora.

For clustering we chose Spectral Clustering [14], an algorithm that we already implemented to identify component structures in source code [15], [16]. In the following subsections the semantic similarity measurement and the clustering are explained in detail.

### A. Computation of Semantic Similarity

To calculate the pairwise similarities between each class or interface of a systems sourcecode we chose an overlap function based on lexical word definitions or glosses as similarity measure, similar to [17].

But how can we find a lexical definition for a class or an interface using Wikipedia? — It is not at all surprising that searching for the class identifier or name does not lead to results in most cases. There are two main reasons for that: on the one hand, many identifiers are special domain specific terms or shortcuts and second, it is common to use a camel case notation for naming classes, parameters or variables etc.. Because of that we introduce the following representation of a class or interface to increase the probability of finding Wikipedia articles.

Each element is represented by a set of words. To take care of the special notation for identifiers a function called *CamelSplit* was implemented. This function generates all substrings of a given word depending on upper-case characters. One special case exist, if an upper-case character is followed by another upper-case character like in the term *"WLANConnectionManager"* the function groups and splits these characters in the following way.

$$\begin{aligned}&\mathit{CamelSplit}(\text{"WLANConnectionManager"}) = \\ &\qquad \{\text{"WLAN"}, \text{"Connection"}, \text{"Manager"}\}\end{aligned}$$

Camel case splitting is a common and standard way of doing textual mining of source code files. The set of words $\Theta_e$ for an element $e$ is defined in formula 1.

$$\Theta_e = \Theta_{\text{Name},e} \bigcup \Theta_{\text{Types,e}} \tag{1}$$

$$\begin{aligned}\Theta_{\text{Name},e} &= n \bigcup \mathit{CamelSplit}(n), \text{ with } n = \mathit{NameOf}(\text{e}) \\ \Theta_{\text{types},e} &= \biguplus_{t \in \mathit{AllTypeOf}(\text{e})} \mathit{NameOf}(t) \bigcup \mathit{CamelSplit}(\mathit{NameOf}(t))\end{aligned}$$

Additional to class or interface names all types and names of parameters or variables, which are used within a source code element were extracted from source code and added to the set including all substrings according to the *CamelSplit* function. The resulting set is a term-document representation of the source code element.

After generating the representation for each element contained in a word set $\Theta$, a Wikipedia search query is performed for each word within this set.

For each source code element all resulting articles are concatenated, so that all of them are now represented by a text of natural language. In the next steps all "uninformative" words like articles, conjunctions, prepositions, etc. are removed from the text using a classical stop-word list [18] for the corresponding language. After stop-word reduction the word frequency of each distinct word is calculated in the remaining text.

For computing the semantic similarity the $n$ most frequented words are taken. In our case studies we choose $n = 150$, because our experiments have shown that this is a good tradeoff between performance and representation variety. The corresponding set of words for an element $e$ is called the Wikipedia representation $X_e$ of the source code element.

The semantic similarity between two source code elements $e_1$ and $e_2$ is defined in formula 2, by computing the overlap of the two word-sets.

$$SemanticSimilarity(e_1, e_2) = \frac{2|X_{e_1} \cap X_{e_2}|}{|X_{e_1}| + |X_{e_2}|} \in [0, 1] \quad (2)$$

## B. Clustering

The computation of pairwise similarities results in a symmetric and weighted graph. In the next step the source code elements should be grouped together, so that elements in the same group are more similar to each other than to those in other groups. This leads to the problems of cutting graphs. In [19] the normalized cut for a graph $G(V, E, w)$ is defined as:

$$\mathit{NCut}(C_1, \ldots, C_k) = min\frac{1}{2}\sum_{i=1}^{k}\frac{f(C_i, V \backslash C_i)}{vol(C_i)} \quad (3)$$

$$\text{with } f(A, B) = \sum_{i \in A, j \in B} w_{ij}$$
$$\text{and } vol(\hat{C}) = \sum_{i \in \hat{C}} deg(v_i)$$

Where *deg* is the degree of a vertex, defined as the number of incident edges to this node. The result of this optimization problem is a set of clusters where each cluster contains a number of nodes with a high similarity between them and a less to the rest of the nodes. Equivalent to cohesion and coupling metrics for component structures in software architecture.

Also in [19] it is proven that the normalized cut problem is NP-hard, so a heuristic is needed in order to calculate results in a reasonable time. One good heuristic to solve this problem is Spectral Clustering [14].

In our case studies we use the normalized Spectral Clustering algorithm as described below.

```
1: Calculate the normalized Laplacian $L_{NOM}$.
2: Determine the first $k$ eigenvectors $e^1, \ldots, e^k$ of $L_{NOM}$.
3: Determine the vectors $x^1, \ldots, x^n \in \mathbb{R}^k$ with $n = |V|$ and $x^i_j = e^j_i$.
4: Cluster the vectors $x^i$ with Neural Gas [20] into $k$ clusters $C_1, \ldots C_k$.
```

As input to perform the algorithm a symmetric and weighted graph $G(V, E, w)$ represented by an adjacency matrix $A_G^{n \times n}$ and the number of clusters $k$ had to be given. A detailed specification of Spectral Clustering is given in [14] and [16].

An open question in most cases using clustering-algorithms is to determine the number of clusters in which the elements should be grouped, if this number is unknown or not given. In the context of Spectral Clustering the eigengap heuristic [14] is used to suggest this number in many cases. The eigengap heuristic is based on searching for gaps within the pairwise differences of the sorted eigenvalues calculated in step two of the algorithm.

## IV. Case Study

To evaluate the presented approach, several case studies were performed with different software systems and discussed with the associated developers and software architects [13].

For this paper we chose a rather small application as extract of these studies, because of the increase in transparency and the reduced complexity. The chosen example shows the main results of our studies.

In the first subsection the example system and the underlaying architecture is described, followed by the presentation of the clustering results of this system. First the approach was applied using typical source code relations and in the next step the semantic similarity relation, introduced in Section III was used. This section is concluded by the comparison of the clustering results.

### A. RASII - the Example System

RASII (a forces information system for fire departments and authorities) contains a collection of applications that collect, distribute and view vital information occurring at a fire-fighting operation and was first presented to public at the CeBIT 2013 fair in Hanover [21]. The application used as case study in this paper is the Group Leader Application (GLApp), which is deployed on an onboard-computer on each fire engine and is composed of several components to provide the following functionality:

The RFID-component realized the identification of fire-fighters by reading the RFID transponder in their clothes with RFID-Readers integrated into the seats of every fire-engine. The sensor developed in this project is also capable to detect persons, because a RFID transponder can be detected, even though no person is present. This might, for example, be the case if a fire fighter left his jacket on the seat.

Based on the person identification and detection the application is able to check if all fire-fighters within the vehicle are sufficiently qualified to handle the actual type of operation. This verification is realized by the application-component and particularly by the GroupLeader-component. The presentation-component enables the group leader to view, manipulate and confirm this information on a 7” touch display built in the vehicle. The collected data is sent to and synchronized with other vehicles or the command control center by the middleware-component. There is also a simulation-component that enables the user to simulate a fictive emergency. This is a feature for testing and demonstration-purposes.

### B. Clustering - Using Source Code Relations

The source code of the Group Leader Application consists of 245 classes and interfaces (all together about 24.000 executable lines of code). Between these 245 source code elements 5352 source code relations exist, 50% of them are Call-Relations. Overall we extract nine different types of relations from source code, which are listed in Table I.

For Spectral Clustering all relations between two elements had to be aggregated to one single edge weight value. To compute this weight we chose the same function as defined

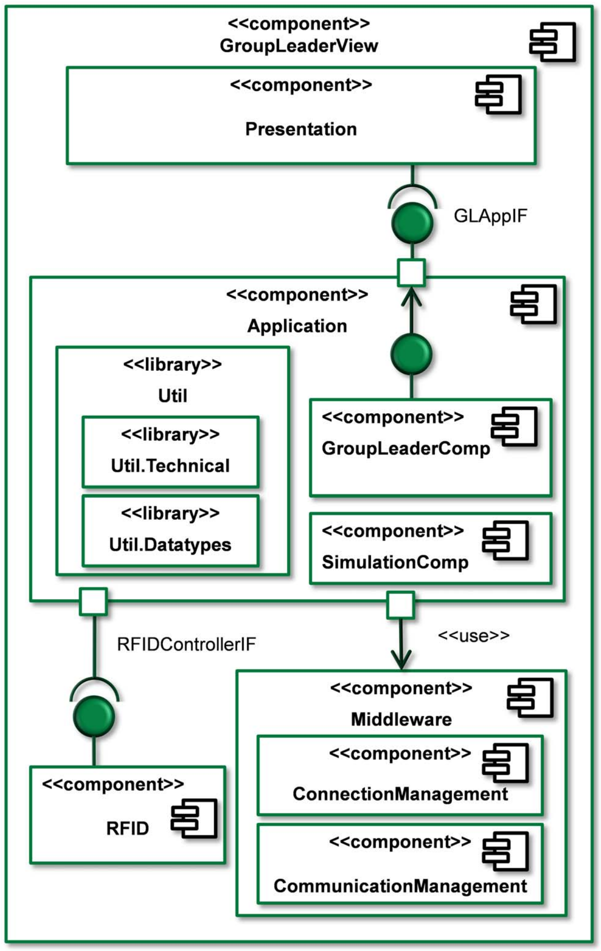

Fig. 1. Component Architecture of Group Leader Application (GLApp).

TABLE I
RELATION TYPES

| Type | Description |
|---|---|
| Implements | A implements B |
| Extends | A inherits B |
| Call | A calls a method on B |
| HasMethodParameter | A uses a method-parameter of type B |
| HasMethodReturn | A uses a return value of type B |
| HasConstructorParameter | A uses a constructor-parameter of type B |
| HasGenericType | B is used as generic type in A |
| Declare | B is declared as member variable in A |
| Instantiate | B is instantiated in A |

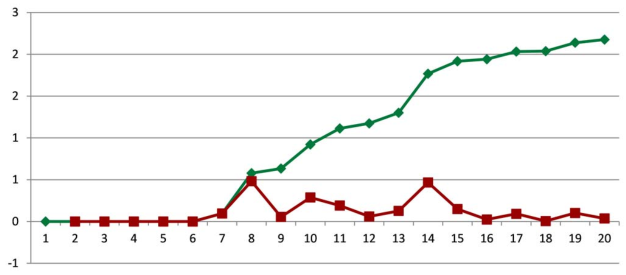

Fig. 2. Sorted eigenvalues (green diamonds) and pairwise differences (red squares) of sourcecode associations.

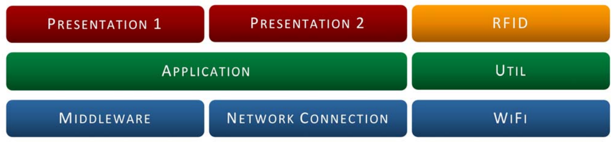

Fig. 3. Clusters generated by using source code relations.

in [15] and [16], which is based on a component model weighting function defining global, type depended weights and the number of relations between the elements.

After calculating the weighted graph the procedure is the same as specified in Section III-B. To determine the number of clusters $k$ the sorted eigenvalues of the normalized graph Laplacian and there pairwise differences were plotted (Fig. 2). According to the eigengap-heuristic (Sec. III-B) we choose $k = 8$, because of the relatively large gap between the seventh and eighth eigenvalue of the graph Laplacian.

The result of the clustering into eight clusters is visualized in Fig. 3. To reduce complexity we chose generic terms for naming the clusters instead of displaying all containing elements. Finding suitable names for clusters is another interesting and difficult problem. Approaches for that could also be found in [13], but we didn't want to address this topic in this paper.

Three clusters representing the network communication were identified, one of them covered all classes which realize the wireless communication and the other ones the classes and interfaces representing the middleware concepts respectively the network connection management. Moreover two clusters containing elements of the presentation layer, a big one with elements of the application layer, one RFID cluster and a small cluster were utility and helper classes are assigned to.

## C. Clustering - Using Semantic Similarity Relations

For the same 245 source code elements, which were analyzed in Section IV-B all 29,890 pairwise similarities were computed as specified in Section III. This results in a weighted graph with edge weights between zero and one. To reduce noise a threshold value of 0.5 was defined. This reduces the number of relevant relations from 29,890 to 814. On the other hand this leads to 34 elements with only irrelevant relations, they were excluded afterwards.

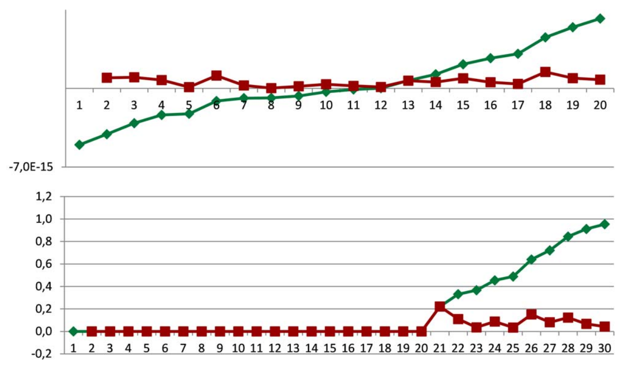

Fig. 4. Sorted eigenvalues (green diamonds) and pairwise differences (red squares) of Similarity associations. The upper plot displays the first twenty eigenvalues with an adjusted scaling.

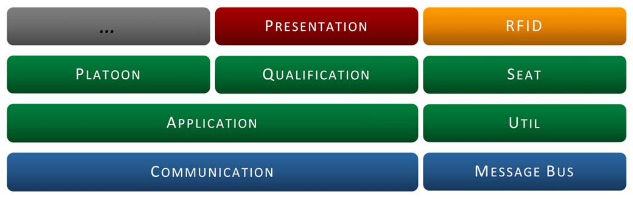

Fig. 5. Clusters generated by using similarity associations.

After that the set of representing words were generated for the remaining 211 source code element resulting in 525 different terms. For each term a Wikipedia query was performed to compute the Wikipedia representation (see Section III) of each element. It is not surprising that only 243 articles were found, because of the typical naming convention within the source code. Nevertheless each class or interface in the inspected system is represented by about four Wikipedia articles.

Regarding the gaps in the sorted eigenvalues of the normalized Laplacian of the similarity graph, shown in figure 4, the eigengap-heuristic points to 7 or 21 as a useful number or clusters, the Spectral Clustering algorithm was executed with both values.

The outcome represents a typical behavior of this algorithm. Grouping the elements in six clusters results in one big and five smaller clusters with about 5 to 10 elements. Dividing the elements into 21 clusters this leads to the partitioning of the big cluster. Furthermore we can observe that by the relatively high number of clusters in relation to the number of elements, a wide range of clusters with few elements represent special aspects of the domain model. The results of the clustering by semantic similarities is visualized in a brief form in figure 5. To simplify the figure only a selection of the clusters is listed.

### D. Comparing Clustering Results

The aim was to group source code elements to cluster to deduce use cases to support software developers or architects in the future. In this subsection the two generated clusterings are compared to identify similarities and differences between the clustering results using typical source code or semantic similarity relations. The construction of the two cluster sets only distinguish in the similarity measure which was used.

It is not surprising that the two cluster sets are not matching exactly, even the number of clusters differs. On a closer consideration we can find in both sets clusters like the *RFID* cluster, in which a majority of the elements are equivalent.

The clusters assigning to the communication components are clustered in a different way. Using source code relations it results in three clusters representing functional aspects, however using semantic relations leads to two clusters representing the communication component in a more conceptual way.

In the presentation layer the partitioning based on source code relations generates two clusters, each representing a view of the application including the corresponding controls. Using semantic similarity these two views were not identified by the algorithm. Instead of two clusters it creates only one cluster which can be assigned exactly to the presentation layer. It contains source code elements like visual controls or elements.

In general, the datatypes of the domain model are assigned to clusters in a different way. Using the semantic measure most of these elements are grouped in separate clusters like the *Seat* or *Qualification* cluster, whereas applying source code relations the datatypes are assigned to clusters which uses these objects at most.

## V. Conclusion and Future Work

In [15] we outline the introduction of further types of associations in the conclusion section. Such a further type of relation can be realized by the introduced semantic similarity relation. In conclusion, clustering source code elements by their semantic similarity leads to a modularization of the software system representing general or technical concepts like RFID-Technology or message bus communication as shown by the case study. But not all domain or application specific concepts were extracted or mentioned. In the presented example application, a snapshot principle is used for data exchange. This concept was not extracted with the presented approach for example.

We can summarize that it can be useful to have a look at the semantic similarity of source code elements, if the analyzed software system is based on a domain model or naming conventions which use terms of a natural language. Nevertheless we had to notice that the semantic similarity analysis taken by itself makes only a small contribution to a better understanding of an unknown software system.

All relevant source code relations exist only in a release version of a software system, not in an early state of a software development process or system extension. So our assumption was to use a textual corpus and the identifiers and abstract types extracted from the source code or other models of the software systems to get relations which are not depending on the state of realization.

At this stage it is not possible to say which clustering is better, because it depends on the use case. Both provide capability to support a developer or software architect in different issues like: "Is the defined component-structure useful? Or in which component could the new functionality be integrated best and what are the resulting effects to the whole system?" The use cases, which could be supported by our approaches have to be clarified in future. This approach can provide new ideas for modularization of a software system and this can be helpful in daily work.

In the future, the following aspects can be interesting for developing tools to support developers or architects that are designing or maintaining software systems:

- Using the semantic similarities between the source code elements for navigation within an integrated development environment (IDE). Which elements are of interest corresponding to the element the developer is actual working on?
- Combination of the source code relation defined in Section IV-B and the semantic similarity relation in one single edge weight value — against the background of getting new indications for modularization.
- Extraction of "hot-spots" by comparing the clustering results automatically and identifying elements with a high semantic similarity and a low code relation weight or vice versa. These spots can be indicators of missing code relations at the actual state of development or inappropriate naming, resulting in an adaption of the domain model alternatively. Certainly it can be assumed that only a small number of them are de facto relevant.

One big issue for further research is to take the history of the development process into account. In both cases, using source code relations or semantic relations as similarity measure the values will change over time. So it would be interesting to analyze the variation of the Wikipedia representation of a single source code element over time for example.

In future studies it is useful to use other systems like open-source systems for evaluation and adaption. But for the first studies it was necessary to have the opportunity to discuss the results with the related developers and architects.

It would be interesting and helpful for evaluation to integrate this approach in an IDE or Case Tool to explore in which parts of the system life-cycle a developer or software architect could be supported.

## REFERENCES

[1] E. Evans, *Domain-driven design: Tackling complexity in the heart of software*. Boston: Addison-Wesley, ©2004.

[2] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[3] N. Anquetil and T. Lethbridge, "Extracting concepts from file names; a new file clustering criterion," in *In Proc. 20th Intl. Conf. Software Engineering*. Press, 1998, pp. 84–93.

[4] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, 2007.

[5] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *15th IEEE International Conference on Program Comprehension (ICPC '07)*, 2007, pp. 37–48.

[6] G. Santos, M. T. Valente, and N. Anquetil, "Remodularization analysis using semantic clustering," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pp. 224–233.

[7] J. I. Maletic and A. Marcus, "Using latent semantic analysis to identify similarities in source code to support program understanding," in *Twelfth Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*, 2000, pp. 46–53.

[8] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. d. Lucia, "Improving software modularization via automated analysis of latent topics and dependencies," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 1, pp. 1–33, 2014.

[9] G. Bavota, A. d. Lucia, A. Marcus, and R. Oliveto, "Using structural and semantic measures to improve software modularization," *Empirical Software Engineering*, vol. 18, no. 5, pp. 901–932, 2013.

[10] L. Kof, *Text Analysis for Requirements Engineering: Application of Computational Linguistics*. Saarbrücken: VDM Verlag Dr. Müller, 2009.

[11] M. Strube and S. P. Ponzetto, "Wikirelate! computing semantic relatedness using wikipedia," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'06. AAAI Press, 2006, pp. 1419–1424. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597348.1597414

[12] K. Lee, H. Kim, H. Shin, and H.-J. Kim, "Folksoviz: A semantic relation-based folksonomy visualization using the wikipedia corpus," in *2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pp. 24–29.

[13] O. Fox, "Einsatz von daten- und text-mining-techniken zur analyse von quellcode," 2014.

[14] U. v. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[15] M. Schindler, C. Deiters, and A. Rausch, "Using spectral clustering to automate identification and optimization of component structures," in *Proceedings of 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 2013, pp. 14–20. [Online]. Available: http://sse-world.de/index.php/forschung/veroeffentlichungen/using-spectral-clustering-to-automate-identification-and-optimiz/

[16] M. Schindler, "Automatische identifikation und optimierung von komponentenstrukturen in softwaresystemen," 2010.

[17] Satanjeev Banerjee, "Extended gloss overlaps as a measure of semantic relatedness," in *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003, pp. 805–810.

[18] A. Rajaraman and J. D. Ullman, "Data mining," in *Mining of Massive Datasets*, A. Rajaraman and J. D. Ullman, Eds. Cambridge: Cambridge University Press, 2011, pp. 1–17.

[19] D. Wagner and F. Wagner, "Between min cut and graph bisection," in *Mathematical Foundations of Computer Science 1993*, ser. Lecture Notes in Computer Science, G. Goos, J. Hartmanis, A. M. Borzyszkowski, and S. Sokołowski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, vol. 711, pp. 744–750.

[20] T. Martinetz and K. Schulten, "A neural-gas network learns topologies," in *Kohonen, Mäkisara et al. (Hg.) 1995 – Artificil Nerual Networks*, pp. 397–402.

[21] Sinosys Ltd. & Co.KG, "Rasii - kräfteinformationssystem," 2013. [Online]. Available: http://rasii.sinosys.de