Using Traceability Links to Assess and Maintain the Quality of Software Documentation

Denys Poshyvanyk, Andrian Marcus[1]
Department of Computer Science
Wayne State University
Detroit MI 48202
denys@cs.wayne.edu, amarcus@cs.wayne.edu

ABSTRACT

The paper proposes an approach for using traceability links to assess and maintain the quality of software documentation. Our position is that quality documentation should accurately reflect the structure of the source code; hence elements of documentation that link to strongly coupled elements of the source code should also be strongly related.

We use latent semantic indexing (LSI) to compute similarities among sections of external documentation. Coupling measures are used to assess the strength of the dependencies among source code elements. The proposed methodology compares the similarities among parts of documentation (e.g., requirements, user manuals) to the coupling among the parts of the source code (e.g., classes) they are linked to.

These measures can be used to improve existing documentation or new one during the evolution of the software.

1. INTRODUCTION

Information retrieval methods, such as Latent Semantic Indexing (LSI) [7], are used in software engineering to identify relationships between elements of documentation, both external and internal. Several successful traceability link recovery methods use these relationships to link source code to external documentations, such as requirements [10], test cases [13], user manuals [6, 15], etc.

External documentation has a structure determined by the authors (e.g., files, sections of documents, directories, etc.). Source code has its own structure determined by the choice of programming language (i.e., files, classes, functions, methods, interfaces, etc.) and design decisions. Traceability link recovery methods link elements of the source code to elements of the documentation. Some methodologies also link elements of the documentation among themselves (e.g., requirements). Information retrieval methods are used by these methodologies to index corpora, where elements of documentation and source code correspond to documents, and similarities among the elements are computed.

Given the nature of source code, its structural elements are also related by control and data flow. Measures such as coupling between classes are often used to express the strength of these dependencies and associations. It is often the case that the external documentation does not reflect these associations. Traceability links directly relate elements of documentation to elements of the source code. We argue that elements of documentation that link to strongly coupled elements of the source code should also be strongly related.

We propose to build on techniques for traceability link recovery based on information retrieval and use the semantic relationships to assess how well the external documentation reflects the structure of the source code. The assumption here is that quality documentation should accurately reflect the structure of the source code. The proposed methodology can be used to improve existing documentation and new one during the evolution of the software, based on this quality criterion.

Specifically, we use LSI to establish relationships between elements of the external documentation and coupling measures to assess the strength of dependency among source code elements (e.g., classes). Strong coupling should be reflected in documentation by strong similarities among corresponding documents. When such similarity is low, it can be strengthen by changes to the existing documentation, such as including comments like "see also class X".

2. PREVIOUS AND RELATED WORK

Our work builds on existing techniques used in traceability link recovery. Relevant to this research is the use of LSI in recovering traceability links between source code and external documentation [6, 13-15]. Common to these approaches is the fact that each approach decomposes external documentation such as requirements, test cases, and manuals into fine grain elements. In parallel, comments and identifiers from source code are extracted and grouped into documents that mirror the structure of the software. LSI is used to index the corpus created from all these documents. Similarities between documents extracted from the source code and those extracted from the external documentation are computed. These similarities are then filtered based on various criteria and used to suggest candidate traceability links, which are usually confirmed by the users.

[1] Corresponding author

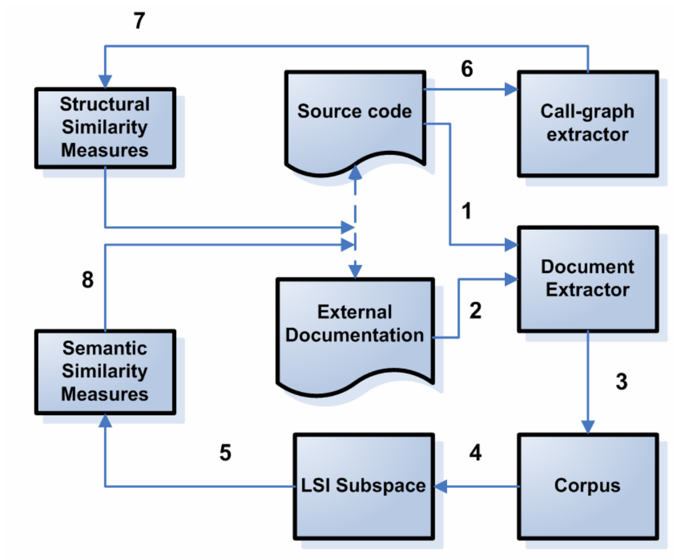

Figure 1. Using semantic and structural information to maintain document quality.

Another use of LSI, relevant to our research, is in establishing relationships between requirements or requirements and design [9, 10].

In addition, LSI is also used to assess the conceptual coupling between classes in Object-Oriented (OO) systems [16]. This conceptual metric, used in conjunction with structural coupling metrics is a good indicator of dependency among elements of the source code (i.e., classes in this case).

3. PROPOSED WORK

The proposed approach extends existing techniques with a set of structural similarity measures to help the developers assess the quality of existing documentation from a certain point of view. More than that, the methodology may be used to improve the documentation as well as help generate new documentation.

Thus, our approach uses additional structural similarities, which can be computed in the form of coupling measures between classes or using call-graph information in case of method level granularity. Technically, in order to suggest related classes using coupling measures they have to be redefined to take into account a pair-wise relations between classes. More details on redefining pair-wise coupling measures can be found in [2]. Alternatively, conceptual coupling measures may be used to suggest conceptually similar classes [16] which do not require additional modifications.

The proposed methodology is based on a set of parallel and sequential steps (see Figure 1), which are partially automated:

1. Extracting documents from source code.
2. Extracting documents from external documentation.
3. Building a corpus.
4. Creating LSI sub-space.
5. Computing semantic similarity measures between elements of the source code and documentation.
6. Computing coupling measures between classes and extracting call-graph of the software.
7. Computing structural similarity measures based on coupling between classes and calling relations between methods.
8. Selecting traceability links between source code and documentation based on structural and semantic similarity measures.
9. Analyzing the similarities between source code elements and documents to infer missing links or cross-references between existing sections of documentation or suggest links for the new documentation and source code.

This process is organized in the pipeline architecture, meaning that the output from one module constitutes the input for the next module.

In the first and the second step, the external documentation and the source code are used to create a corpus that is used to generate the semantic space for information retrieval (see step 3 and 4).

The semantic space, named the LSI subspace, is automatically generated in phases (4). This step is based on the LSI mechanism with more details on which can be found in [14, 15]. Once the LSI subspace is constructed, each part of the documentation and each source code element will be represented as a vector in this space. Based on this representation, a semantic similarity measure is defined (see section Figure 1). The measure is further used to identify elements of the source code that relate closely to a given part of the documentation or vice-versa. In addition, coupling measures and call-graph are extracted from the source code and structural similarities between classes and methods are identified. The structural and semantic similarity measures are automatically computed, while the user selects the appropriate pairs of documents that correspond to traceability links. The final phase of the process consists of analyzing structural similarities among source code elements and semantic similarities among sections in documentation and missing links are suggested.

4. DEFINING SIMILARITIES

We provide more details on our approach with some background definitions, necessary to understand the approach.

4.1 Semantic Similarity Measure

Definition. A *source code document* (or simply document) d is any contiguous set of lines of source code and/or text. A document is a file of source code, implementation of a class or method, function.

Definition. An *external document e* is any contiguous set of lines of text from external documentation (i.e., manual, design documentation, requirement documents, test suites, etc.). Typically an external document is a section, a chapter, or maybe an entire file of text.

Definition. The external documentation is also a set of documents $E = \{e_1, e_2, \ldots, e_m\}$. The total number of documents in the documentation is $m = |E|$

Definition. The source code is also a set of documents $D = \{d_1, d_2, \ldots, d_m\}$. The total number of documents in the documentation is $n = |D|$.

Definition. A *software system* is a set of documents (source code and external)

$S = D \cup E = \{d_1, d_2, \ldots, d_n\} \cup \{e_1, e_2, \ldots, e_m\}$. The total number of documents in the system is $n+ m = |S|$.

Definition. A *file* f_i, is then composed of a number of documents and the union of all files is S. Size of a file, f_i, is the number of documents in the file, noted $|f_i|$.

LSI uses the set $S = \{d_1, d_2, \ldots, d_n, e_1, e_2, \ldots, e_m\}$ as input and determines the *vocabulary V* of the corpus. The number of words (or terms) in the vocabulary is $v = |V|$. Based on the frequency of the occurrence of the terms in the documents and in the entire collection, each term is weighted with a combination of a local log weight and a global entropy weight. A term-document matrix $\mathbf{X} \in \boldsymbol{R}^{v\times n}$ is constructed. Based on the user-selected dimensionality (k), SVD creates the LSI subspace. The term-document matrix is then projected onto the k-dimensional LSI subspace. Each document $d_i \in D$ will correspond to a vector $\mathbf{x_i} \in \mathbf{X}$ projected onto the LSI subspace.

Definition. For two documents d_i and d_j, the *semantic similarity* between them is measured by the cosine between their corresponding vectors $semantic\_sim(d_i, d_j) = \cos(\mathbf{x_i}, \mathbf{y_i})$ The value of the measure will be between [-1, 1] with value (almost) 1 representing that the two are (almost) identical.

4.2 Structural Similarity Measure

Definition. For two documents d_i and d_j, , which belong to the source code, the *structural similarity* is measured as one of the coupling measures in case of class level granularity or presence of called-caller relation in terms of methods level granularity. We define this measure as $structural\_sim(d_i, d_j)$.

For class level granularity we may use variety of coupling measures, e.g. CBO (coupling between objects) [4, 5], RFC (response for class) [4] and RFC_∞ [5], MPC (message passing coupling) [12], DAC (data abstraction coupling) [12], ICP (information-flow-based coupling) [11], the suite of coupling measures by Briand et al. (IFCAIC, ACAIC, OCAIC, FCAEC, etc) [3], Ce (efferent coupling), Ca (afferent coupling), COF (coupling factor)[2], CoCC (conceptual coupling between classes) [16].

In this step of the process, the similarities between each pair of documents from $E \times D$ are computed and ranked, based on the above definitions. It is an automated step, done without user intervention.

4.3 Maintaining Quality of Documentation

With the similarity measures computed in the previous step, the user can initiate the analysis of links between sections of documentation based on the structural similarities computed from the source code. For a given class in the source code d_i the system will return the most-similar source-code documents d_j, etc based on $structural\_sim(d_i, d_j)$. If the traceability link between sections of documentation which describe d_i and d_j is missing, then it will be suggested by the system.

Although it is the user's task to verify the validity of the link suggested by the system. In some cases, part of the documentation may refer to more than one source-code document, or a source-code document may be described by more than one external document. In such cases, the user needs to investigate the next suggested document. Since the process is only partially automated, the stopping criterion is defined by the user, once all the links relevant to a query document are retrieved. The process can be used on a single query document or multiple ones, essentially for the entire system under analysis.

To operate at system level, the user has two options. One is to determine a threshold ε for the similarity measure that identifies which documents are considered "linked". In other words, among all the pairs from $E \times D$, only those will be suggested that have a semantic similarity measure greater than ε. Similarly, for structural similarity measure we may apply threshold on the values of coupling measures which can be used to suggest similar classes or methods. The threshold is determined empirically and varies from corpus to corpus. The issue of the "best" threshold for this type of corpus (i.e., combining source code and documentation) is still open and further research is needed. Many IR methods (especially search engines) use this approach, where the retrieved documents are ranked by the "relevancy" to a query.

An alternative option for the user is simply to retrieve the top θ ranked links for each document, where $\theta \in \{1, 2, \ldots, n\}$. In this case, a threshold on the number of recovered links, regardless of the actual value of the similarity measure, is imposed. This approach works equally well for structural and semantic similarity measures.

Finally, the user can opt to combine the two types of thresholds, for example to retrieve the top θ ranked links among those that have a similarity measure greater-than ε. The different choices accommodate different user needs. Using the threshold method, with a high enough threshold value will allow the system to suggest few false positives. Too high of a threshold will result in missing relevant links. Using the ranking method, the user will retrieve more relevant links at the expense of yielding more false positives.

5. PRELIMINARY EVALUATION

We have conducted several preliminary studies to evaluate our approach. The goal was to assess how well structural information, obtained by program analysis on the source code, can improve the structure of external documentation in addition to traceability links retrieved by LSI.

In our preliminary study we used LEDA software system, Library of Efficient Data types and Algorithms, which we previously used in our original LSI-based approach for traceability link recovery [14, 15].

As in the previous study on LEDA, we followed the same steps: we used the entire user manuals and source code to ensure the generation of a rich LSI-based semantic space and resulting vocabulary.

We have extracted LEDA call-graphs and coupling measures using Columbus [8] tool. Conceptual coupling measures for LEDA source code have been extracted using our IRC^{3}M tool [16]. We have computed and used the following coupling measures: CBO, RFC, MPC, DAC, ICP, ACAIC, OCAIC, ACMIC, OCMIC, CoCC.

Based on our preliminary evaluation we identified several additional links between manuals in external documentation which did not have explicit links between them, although classes to which they were connected were strongly coupled. Overall, coupling measures seem to suggest relevant links between sections in documentation, which we believe should increase recall values of existing approaches for maintaining traceability links. However, these conclusions are preliminary and should be drawn based on complete case studies.

Since we did not recover all traceability links between all classes and sections in documentation, we could not compute precision and recall values and directly compare these results with the previous results in [1, 15]. However, this remains our future work. In addition to this we will recover all traceability links using our approach and compare recall and precision for our approach with all input parameters, such as different threshold levels and particular thresholds for structural coupling measures etc.

6. CONCLUSIONS

The proposed work uses state of the art approaches in traceability link recovery and program analysis. It extends this work to address how well external documentation reflects the structure of the source code.

A prototype tool that implements this methodology is under development, as an Eclipse plug-in. A set of extensive case studies are being designed in order to evaluate the proposed approach.

7. ACKNOWLEDGMENTS

This research is supported in part by a grant from the National Science Foundation (CCF-0438970) and a 2006 IBM Eclipse Innovation Award.

8. REFERENCES

[1] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E., "Recovering Traceability Links between Code and Documentation", IEEE Transactions on Software Engineering, vol. 28, no. 10, October 2002, pp. 970 - 983.

[2] Briand, L. C., Daly, J., and Wüst, J., "A Unified Framework for Coupling Measurement in Object Oriented Systems", IEEE TSE, vol. 25, no. 1, January 1999, pp. 91-121.

[3] Briand, L. C., Devanbu, P., and Melo, W. L., "An investigation into coupling measures for C++", in Proc. of International. Conf. on Software Engineering,1997,pp.412- 421.

[4] Chidamber, S. R. and Kemerer, C. F., "Towards a Metrics Suite for Object Oriented Design", in Proceedings of Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'91), 1991, pp. 197-211.

[5] Chidamber, S. R. and Kemerer, C. F., "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994, pp. 476-493.

[6] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G., "Enhancing an Artefact Management System with Traceability Recovery Features", in Proc. of ICSM, 2004, pp. 306-315.

[7] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R., "Indexing by Latent Semantic Analysis", Journal of the American Society for Information Science, vol. 41, 1990, pp. 391-407.

[8] Ferenc, R., Siket, I., and Gyimóthy, T., "Extracting facts from open source software", in Proc. of 20th International Conference on Software Maintenance, 2004, pp. 60-69.

[9] Hayes, J. H., Dekhtyar, A., and Osborne, J., "Improving Requirements Tracebility via Information Retrieval", in Proc. of IEEE Int. Requirements Engineering Conf., 2003, pp. 138-147.

[10] Hayes, J. H., Dekhtyar, A., and Sundaram, S. K., "Advancing candidate link generation for requirements tracing: the study of methods", IEEE Transactions on Software Engineering, vol. 32, no. 1, January 2006 2006, pp. 4-19.

[11] Lee, Y. S., Liang, B. S., Wu, S. F., and Wang, F. J., "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow", in Proc. of International Conference on Software Quality, Maribor, Slovenia, 1995.

[12] Li, W. and Henry, S., "Object-oriented metrics that predict maintainability", Journal of Systems and Software, vol. 23, no. 2, 1993, pp. 111-122.

[13] Lormans, M. and Van Deursen, A., "Can LSI help Reconstructing Requirements Traceability in Design and Test?" in Proc. of European CSMR, 2006, pp. 47-56.

[14] Marcus, A. and Maletic, J. I., "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", in Proc. ICSE, May 3-10 2003, pp. 125-137.

[15] Marcus, A., Maletic, J. I., and Sergeyev, A., "Recovery of Traceability Links Between Software Documentation and Source Code", Int. Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 4, Oct. 2005, pp. 811-836.

[16] Poshyvanyk, D. and Marcus, A., "The Conceptual Coupling Metrics for Object-Oriented Systems", in Proc. of 22nd IEEE ICSM, Sept 2006, pp. 469 - 478.