Using Information Retrieval to Support Software Maintenance Tasks

Denys Poshyvanyk

Computer Science Department
The College of William and Mary
Williamsburg, VA 23185
denys@cs.wm.edu

Abstract

This paper presents an approach based on Information Retrieval (IR) techniques for extracting and representing the unstructured information in large software systems such that it can be automatically combined with analysis of program dependencies and execution traces to define new techniques for feature location, impact analysis, and software measurement tasks. We expect that these new techniques will contribute directly to the improvement of design of incremental changes and thus increased software quality and reduction of software maintenance costs. The presented results are based on the author's doctoral dissertation [23].

1. Introduction

Software is comprised of a multitude of artifacts; some of them are intended to be read by the compiler, while many others are intended to be read by developers. During software evolution, developers have to maintain large software systems often written by others. The developer-centric information is often expressed in natural language and it is embedded in documentation and source code. External documentation written in natural language (e.g., requirements, design documents, user manuals, etc.), the comments, and the identifiers encode the domain of the software and capture design decisions, change requests, developer information, etc. This unstructured information is usually larger in size than the source code. Efficient mechanisms for storing and sharing this information are needed, especially when development teams are distributed geographically and change frequently over time. Given the large amount of textual data present in existing software systems, tools are necessary for its storage, retrieval, and analysis before it is delivered to the users.

The dissertation proposes and validates the use of Information Retrieval techniques to extract, represent, and analyze textual information in large scale software systems such that it can be automatically combined with structural information to better support a variety of software maintenance tasks and activities. Specifically, the research focuses on combining textual information extracted with IR methods with program dependencies, execution traces, and other analyses to define new techniques for feature location, impact analysis, and new measures for class cohesion and coupling.

The main contributions of this dissertation are [23]:

- Definition and validation of an approach to concept location in source code, which combines Formal Concept Analysis (FCA) and Latent Semantic Indexing (LSI) [30];
- Definition and validation of a feature location approach which relies on the combination of probabilistic ranking of methods based on execution scenarios and IR [25, 26];
- Definition and validation of a semi-automated technique for feature location in source code based on the combination of a single execution trace and comments and identifiers from source code [14];
- Definition and validation of new measures for cohesion [18] and coupling [27] of classes in Object-Oriented systems based on the analysis of textual information in software. The cohesion metrics are applied for identifying fault-prone classes in large open-source systems [19], whereas the coupling metrics are used to support impact analysis tasks [29].

We expect that these new techniques and software measures will contribute directly to the improvement of the design of incremental changes of software and thus will lead to increased software quality and reduced software maintenance costs.

2. Motivation and Background

Identifiers chosen by programmers as names for classes, methods, or attributes contain valuable information and account for more than a half of the source code content in existing software systems [8]. These names often serve as a starting point in many

program comprehension tasks [5], thus it is important that these names clearly reflect the concepts that they are supposed to represent as self-documenting identifiers reduce the time and effort necessary to obtain necessary comprehension level for the software maintenance task at hand [3]. The problem of extracting and analyzing the textual information in software artifacts was recognized by researchers about two decades ago. IR methods were proposed and used successfully to accomplish these tasks. Early models were used to construct software libraries and support reuse tasks, while more recent work focused on specific software maintenance and development tasks such as feature location and traceability link recovery.

Several approaches have been developed to recover traceability links between source code and external documentation using probabilistic IR, vector space models [1, 9] and LSI [7, 9, 17]. Other work proposed a set of approaches to recover traceability links among requirements [9], requirements and source code [1, 9, 17], and requirements and test cases [15]. A set of tools that integrates facilities to manage traceability links among different types of software artifacts was developed and evaluated recently [7].

IR methods have been also successfully used for concept and feature location [14, 20, 26, 30] in source code. Other approaches use IR methods to classify software systems based on source code in open-source repositories [11] as well as to cluster source code to obtain high-level views of software systems [12, 16]. IR techniques were also used to identify the starting impact set of a maintenance request [2, 6] and to link change request descriptions to sets of historical file revisions impacted by similar past change requests [4].

IR approaches have also been used in the context of software measurement to assess the quality of identifiers and comments [13], conceptual cohesion [18, 19] and coupling [27, 29] of classes, as well as assessing and maintaining the quality of external software documentation [28]. In addition, IR techniques have been applied to several other tasks in the past few years, such as bug fix assignment based on problem description reports, identification of duplicate bug reports [33], estimating the time to fix a particular bug based on similar bug reports, classification of software maintenance requests, providing recommendations for novice programmers, identifying developer contributions, mining concept keywords, identification of changes from software repositories, and finding similar software applications. Due do space limitations, we omitted some of the citations in this section, whereas the complete related work can be found in [23].

These IR based approaches to software engineering problems differ not only in their scope, but also in their underlying indexing mechanism, corpus construction, and data analysis method. A general model for using IR methods can be described with the following steps:

- **Preparing the corpus**. A corpus is created using the source code and other linguistic software artifacts, such as the external documentation. Various pre-processing methods are employed in the corpus construction, some based on natural language processing techniques, such as word stemming or external ontologies. Each document in the corpus corresponds to a specific software element, such as a file, a class, or a method.
- **Indexing the corpus**. An IR method is used to index the corpus, such as vector space models, Latent Semantic Indexing, naïve Bayes classifiers, or other probabilistic models, etc. A semantic space of the software system is created.
- **Computing similarities**. A similarity measure between the documents in the corpus is defined and similarities are computed among the corresponding software elements. These measures are commonly referred to as semantic similarities.
- **Solving software maintenance tasks**. The semantic similarities are used to solve the maintenance or development task at hand. Some approaches combine these measures with additional data extracted with structural software analysis tools, such as program dependencies, software change data, execution traces, etc.

All this body of work (most of it done in the past five-seven years) shows the usefulness of the IR based approaches to support software engineering tasks, but also highlights limitations and helps define research directions in the field. We outline the contributions of this dissertation to the field in the next section.

3. Research Contributions

We propose the use of IR techniques to extract and represent the semantic information in large scale software systems such that it can be automatically combined with structural information to better support concept and feature location in source code, impact analysis, and software measurement tasks. Specifically, the research in this dissertation focuses on combining IR-based analysis data with the analysis of program dependencies, execution traces to define new techniques for feature location, impact analysis, and software measurement.

3.1. Concept Location with Concept Lattices

We have developed an approach to concept location in source code which combines Formal Concept Analysis and Latent Semantic Indexing [30]. In this approach, LSI is used to map the concepts expressed in

queries written by programmers to relevant parts of the source code, presented as a ranked list of search results. Given the ranked list of source code elements, the approach selects the most relevant attributes from these documents and organizes the results in a concept lattice generated via FCA. The approach is evaluated in a case study on concept location in the source code of Eclipse, an industrial-size integrated development environment. The results of the case study indicate that the proposed approach is effective in organizing different concepts and their relationships present in the subset of the search results. The proposed concept location method outperforms the simple ranking of the search results, reducing programmers' effort.

3.2. PROMESIR

We have developed an approach for feature location using probabilistic ranking of methods based on execution scenarios and IR, namely **P**robabilistic **R**anking **O**f **M**ethods based on **E**xecution **S**cenarios and **I**nformation **R**etrieval (PROMESIR) [25, 26]. In this work, we recast the problem of feature location in source code as decision-making problem in the presence of uncertainty. The solution to the problem is formulated as a combination of the opinions of different experts. The experts in this work are two existing techniques for feature location: a scenario-based probabilistic ranking of events and an IR-based technique that uses LSI. We have empirically evaluated this combination of the experts through several case studies which use the source code of the Mozilla Web browser and the Eclipse integrated development environment. The results show that the combination of experts significantly improves the effectiveness of feature location when compared to each of the experts used independently.

3.3. SITIR

We have implemented a semi-automated technique for feature location in source code named SITIR which is based on combining information from two different sources: an execution trace and the comments and identifiers from source code [14]. Using this technique, users execute a single scenario (or a part of it) which exercises the desired feature and all executed methods are identified based on the collected trace. The source code is indexed using LSI, which allows users to write queries relevant to the desired feature and rank all the executed methods based on their textual similarity to the query. Two case studies on open source software, JEdit and Eclipse, indicate that the new technique has accuracy comparable with previously published approaches such as PROMESIR and it is easier to use as it considerably simplifies collecting execution traces.

3.4. The conceptual cohesion of classes

We proposed a new measure for cohesion of classes in an OO software system based on the analysis of unstructured information embedded in source code, such as comments and identifiers [18]. The measure, named the Conceptual Cohesion of Classes (C3), is inspired by the mechanisms used to measure textual coherence in cognitive psychology and computational linguistics. We have devised the principles and the technology that stand behind the C3 measure. A large case study on three open source software systems is presented, which compares the new measure with an extensive set of existing metrics. The case study shows that the proposed measure captures different aspects of class cohesion from any of the existing cohesions measures. In addition, combining C3 with existing structural cohesion metrics proves to be a better predictor for fault proneness of classes when compared to different combinations of structural cohesion metrics [19].

3.5. The conceptual coupling of classes

We have presented a new set of coupling measures for OO systems – named conceptual coupling, based on the semantic information encoded in identifiers and comments obtained from source code, [27]. Conceptual coupling is based on measuring the degree to which the identifiers and comments from different classes relate to each other. This type of relationship is measured through the use of Information Retrieval techniques. The proposed measures are different from existing coupling measures, and they capture new dimensions of coupling which are not captured by the existing coupling measures. The case study also investigates the use of the conceptual coupling measures during change impact analysis. We report the findings of a case study on Mozilla, where the conceptual coupling metrics were compared to nine existing structural coupling metrics and proved to be better predictors of classes impacted by changes [29].

4. Publications and Current Work

Several refereed publications resulted from this research [14, 18, 19, 26, 27, 29-31] in addition to the dissertation [23].

The current work continues today on improving existing techniques for feature location, impact analysis, traceability link recovery, software reuse, mining developer expertise, and already resulted in several refereed conference publications [10, 21, 22, 24, 32].

5. Acknowledgements

I would like to thank my mentor, Dr. Andrian Marcus, without his support this thesis would not have been possible. I would also like to acknowledge Dr. Václav Rajlich and Dr. Jonathan Maletic for their endless feedback and support on this research. I am grateful to Dr. Giuliano Antoniol and Dr. Yann-Gaël Guéhéneuc for their strong collaboration and support on several research projects including PROMESIR [25, 26]. I acknowledge Dr. Rudolf Ferenc and Dr. Tibor Gyimóthy for their support in empirical studies [19, 29]. The doctoral thesis was supported in part by the grants from the United States National Science Foundation CCF-0438970 and CCF-0820133. The current work is supported in part by a grant from the United States Air Force Office of Scientific Research under grant number FA9550-07-1-0030.

6. References

[1] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E., "Recovering Traceability Links between Code and Documentation", *IEEE TSE*, 28/10, October 2002, pp. 970 - 983.

[2] Antoniol, G., Canfora, G., Casazza, G., and Lucia, A., "Identifying the Starting Impact Set of a Maintenance Request: A Case Study", in Proc. of 4th CSMR'00, Feb. 2000, pp. 227-231.

[3] Antoniol, G., Gueheneuc, Y.-G., Merlo, E., and Tonella, P., "Mining the Lexicon Used by Programmers during Software Evolution", in Proc. of 23rd ICSM'07, 2007, pp. 14-23.

[4] Canfora, G. and Cerulo, L., "Impact Analysis by Mining Software and Change Request Repositories", in Proc. of 11th METRICS'05, Sept. 19-22 2005, pp. 20-29.

[5] Caprile, C. and Tonella, P., "Nomen Est Omen: Analyzing the Language of Function Identifiers", in Proc. of 6th WCRE'99, October 1999, pp. 112-122.

[6] Cubranic, D., Murphy, G. C., Singer, J., and Booth, K. S., "Hipikat: A Project Memory for Software Development", *IEEE TSE*, vol. 31, no. 6, June 2005, pp. 446-465.

[7] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G., "Recovering Traceability Links in Software Artefact Management Systems", *ACM TOSEM*, vol. 16, no. 4, 2007.

[8] Deissenboeck, F. and Pizka , M., "Concise and Consistent Naming", *Software Quality Journal*, vol. 14, no. 3, 2006, pp. 261-282

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

[10] Kagdi, H. and Poshyvanyk, D., "Who Can Help Me with this Change Request?" in Proc. of 17th ICPC'09, Canada, 2009.

[11] Kawaguchi, S., Garg, P. K., Matsushita, M., and Inoue, K., "MUDABlue: An automatic categorization system for Open Source repositories", *Journal of Systems and Software*, vol. 79, no. 7, 2006, pp. 939-953.

[12] Kuhn, A., Ducasse, S., and Gîrba, T., "Semantic Clustering: Identifying Topics in Source Code", *Information and Software Technology*, vol. 49, no. 3, March 2007, pp. 230-243.

[13] Lawrie, D., Feild, H., and Binkley, D., "Leveraged Quality Assessment Using Information Retrieval Techniques", in 14th ICPC'06, Athens, Greece, June 14-16 2006, pp. 149-158.

[14] Liu, D., Marcus, A., Poshyvanyk, D., and Rajlich, V., "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", in Proc. of 22nd ASE'07, Atlanta, Georgia, November 5-9 2007, pp. 234-243.

[15] Lormans, M. and Van Deursen, A., "Can LSI help Reconstructing Requirements Traceability in Design and Test?" in Proc. of 10th CSMR'06, March 2006, pp. 47-56.

[16] Maletic, J. I. and Marcus, A., "Supporting Program Comprehension Using Semantic and Structural Information", in Proc. of 23rd ICSE'01, Canada, May 12-19 2001, pp. 103-112.

[17] Marcus, A., Maletic, J. I., and Sergeyev, A., "Recovery of Traceability Links Between Software Documentation and Source Code", *International Journal of Software Engineering and Knowledge Engineering*, vol. 15/4, October 2005, pp. 811-836.

[18] Marcus, A. and Poshyvanyk, D., "The Conceptual Cohesion of Classes", in Proc. of 21st ICSM'05, Sept. 25-30 2005, pp. 133-142.

[19] Marcus, A., Poshyvanyk, D., and Ferenc, R., "Using the Conceptual Cohesion of Classes for Fault Prediction in Object Oriented Systems", *IEEE TSE*, vol. 34, no. 2, 2008, pp. 287-300.

[20] Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J., "An Information Retrieval Approach to Concept Location in Source Code", in Proc. of 11th WCRE'04, Nov. 9-12 2004, pp. 214-223.

[21] McMillan, C., Poshyvanyk, D., and Revelle, M., "Combining Textual and Structural Analysis of Software Artifacts for Traceability Link Recovery", in Proc. of TEFSE'09, Canada, 2009.

[22] Pierret, D. and Poshyvanyk, D., "An Empirical Exploration of Regularities in Open-Source Software Lexicons", in Proc. of 17th ICPC'09, Vancouver, British Columbia, Canada, 2009.

[23] Poshyvanyk, D., *Using Information Retrieval to Support Software Maintenance Tasks*, Wayne State University, Detroit, MI, USA, Doctoral, 2008.

[24] Poshyvanyk, D. and Grechanik, M., "Creating and Evolving Software by Searching, Selecting and Synthesizing Relevant Source Code", in Proc. of 31st ICSE'09, Canada, May 2009.

[25] Poshyvanyk, D., Guéhéneuc, Y. G., Marcus, A., Antoniol, G., and Rajlich, V., "Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification", in Proc. of 14th ICPC'06, Athens, Greece, 2006, pp. 137-146.

[26] Poshyvanyk, D., Guéhéneuc, Y. G., Marcus, A., Antoniol, G., and Rajlich, V., "Feature Location using Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval", *IEEE TSE*, vol. 33, no. 6, June 2007, pp. 420-432.

[27] Poshyvanyk, D. and Marcus, A., "The Conceptual Coupling Metrics for Object-Oriented Systems", in Proc. of 22nd ICSM'06, Philadelphia, PA, September 25-27 2006, pp. 469 - 478.

[28] Poshyvanyk, D. and Marcus, A., "Using Traceability Links to Assess and Maintain the Quality of Software Documentation", in Proc. of TEFSE'07, 2007, pp. 27-30.

[29] Poshyvanyk, D., Marcus, A., Ferenc, R., and Gyimóthy, T., "Using Information Retrieval based Coupling Measures for Impact Analysis", *Empirical Software Engineering*, vol. 14, no. 1, 2009, pp. 5-32.

[30] Poshyvanyk, D. and Marcus, D., "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code", in Proc. of 15th ICPC'07, June 2007, pp. 37-48.

[31] Poshyvanyk, D., Petrenko, M., Marcus, A., Xie, X., and Liu, D., "Source Code Exploration with Google ", in Proc. of 22nd ICSM'06, Philadelphia, PA, 2006, pp. 334 - 338.

[32] Revelle, M. and Poshyvanyk, D., "An Exploratory Study on Assessing Feature Location Techniques", in Proc. of 17th ICPC'09, Canada, May 17-19 2009.

[33] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J., "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information", in Proc. of 30th ICSE'08, Leipzig, Germany, May 10 - 18 2008, pp. 461-470.