POSITION PAPER: IMPROVING SOURCE CODE REUSE THROUGH DOCUMENTATION STANDARDIZATION

Rubén Álvarez-González, Sonia Sanchez-Cuadrado, Héctor García

Abstract: In the context of Software Reuse providing techniques to support source code retrieval has been widely experimented. However, much effort is required in order to find how to match classical Information Retrieval and source code characteristics and implicit information. Introducing linguistic theories in the software development process, in terms of documentation standardization may produce significant benefits when applying Information Retrieval techniques. The goal of our research is to provide a tool to improve source code search and retrieval In order to achieve this goal we apply some linguistic rules to the development process.

Keywords: Software Reuse, Information Retrieval, source code documentation.

ACM Classification Keywords: Reusable software, Information theory, Documentation.

Conference: The paper is selected from Sixth International Conference on Information Research and Applications – i.Tech 2008, Varna, Bulgaria, June-July 2008

Introduction

In the area of Software Reuse research projects have obtained promising results since the decade of 90's [Prieto-Díaz, 1991]. Some major concerns are reducing effort and cost, increasing competitiveness, through reusing documents, models or source code. The first steps in Software Reuse consisted of reusing source code. Since then, researchers have been increasing continuously the possibilities of reuse. However much work is still required to provide proper methods, techniques and repositories allowing source code reuse.

As long as source code is basically text, the application of Information Retrieval (IR) techniques is a common approach. These techniques have proven its usefulness for purposes such as document retrieval. The similarities between text documents and source code make IR a good candidate to support source code reuse.

However the application of IR techniques to source code retrieval is not exempt of troubles. Reserved words of languages are to be considered as empty words in IR, while some of them could be important in Natural Language Processing. Programmers may define meaningless identifiers that are considered correct in the context of compiler theory; however this does not help in applying IR techniques. The difficulties found while automating semantics processing of source code are also a matter of fact.

In our research we are proposing a tool to ease retrieving source code (e.g. classes) from a repository. As long as many management applications are developed under the Object Oriented paradigm we chose to focus on it, particularly the .NET framework.

The purpose was to define a repository to store the source code from developed applications, already tested, after checking that they are consistent enough to be reused. A search engine is to be deployed to allow searching the repository. IR is applied to comments, once they have been written compliant to the standard we have defined. Also we focus the search in source code (e.g. class name, method signature). The purpose is to provide the users the capability to find non exact matches, but to retrieve variations on the query terms (i.e. words with similar morphology). Finally, ranking techniques are applied to the results.

Related work

Some applications have been deployed in order to provide to users source code retrieval capabilities. Swik or Google Code Search are applications oriented to find full open source code to Internet users. As long as these applications are focused on the community it is difficult to use them within an organization where sharing code is not an option.

The work by Sindhgatta [Sindhgatta, 2006], named JSearch, provides a plug-in for Eclipse development environment. The goal is to provide programmers the location of source code that may be used as an example, useful while developing new applications. The source code assets are already developed libraries or classes, in the context of the same organization. However this may not be considered as Software Reuse.

JSearch applies IR techniques, modified for the specific purpose of source code retrieval. The tool provides a source code transformation module. This module extracts and modifies some features in order to properly model and index a class or library. It is focused on users that know exactly what they look for, but have little knowledge on how to use them. This is to say, it finds code assets that use a specific class or method. Then, if a user looks for functionality instead for a similar example the system returns unexpected results.

The same situation is found in the results obtained by [Krugle, 2008] and [Koders, 2008]. Both tools assume that the programmer have deep knowledge of the structure of the source code to be located.

In a more general context, when the exact name of a class or method to be found is not known, comments may be useful to override the problem. As long as comments should describe briefly the functionality provided by a source code asset they can be used to understand source code. The main concern is that it can not be assumed that comments are correct or represent properly the functionality. This fact affects significantly to the goodness of obtained results.

The work by Ying, Wright and Abrams [TT Ying, 2005] describes a feasible classification for the different types of comments in source code. In the case of source code search engines described above the comments such as “do not delete this line or execution shall produce errors” are considered as proper comments. It will not help in obtaining appropriated results. This is why good comments are a must if it is desired to provide support for Software Reuse.

Due to the limited IR capabilities of the tools available at Internet we described above we evaluated [Lucene, 2008] and [Google Desktop, 2008]. The purpose of the evaluation was to infer which third-party IR capabilities may be reused for the purpose of source code reuse. We chose these tools because they can be used in both local and centralized environments, and because we did not incurred in licence fees.

These tools provide the capability of finding documents containing syntactically close to those in query criteria. The test consisted on designing a corpus that consisted on a set of source code documents. After indexing the corpus three different query types were executed:

- Queries with only one search term that is located, at least, in an indexed document. The purpose is to check if the results contain documents with exact term matching.
- Queries with a term containing the same root of the first query (e.g. plurals, derived words). The purpose is to check if the results of the first query are included in the results of the second.
- Queries with shorter forms of those in the first query. The purpose is to check the deviation on the results derived from loosing specificity.

With Google Desktop we needed to use only two Spanish terms: “entrada” and “entradas” (in English “input” and “inputs”). With de first term the search engine found one document. But with the second term, it could not found it. This, drove us to conclude that Google Desktop looks for exact term matching.

The same results were obtained during the tests for Lucene. Also we used two terms, in this case “Developer” and “Developers”…

Software reuse, documentation and linguistic theories

In this section we describe the tool we are proposing in order to improve the results of IR applied to source code retrieval. First we will show the architecture of the tool, and then we are to analyze the characteristics of a high quality comment. Finally we suggest some techniques which application may lead to the improvement of the results.

The tool is intended to receive, from programmers, a text file containing the definition of one or more classes. Once the file has been received each class is separated from the rest in the same file. Each class is processed separately, obtaining a model of the class in the form of a digest document. Information retrieval techniques are applied to these documents. The information in documents contains:

- Class name
- Classes in import clauses
- Names of the inherited classes
- A digest from method signature
- Comments from source code

Comments to be considered have been reduced in order to improve the results for queries looking for functionality instead for specific resource usage. For such a purpose selected comments are those compliant to .NET style from Visual Studio 2005. Figure 1 shows the structure of a source code asset. Imports, identifiers and selected comments have been highlighted.

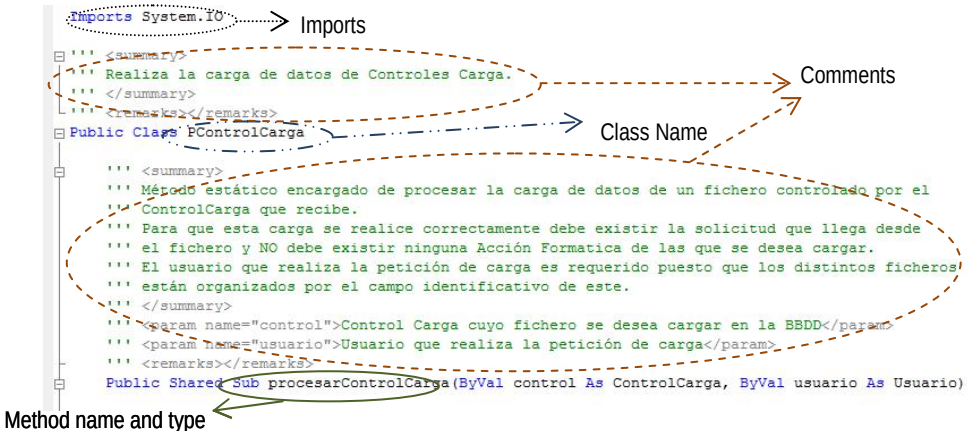

Figure 1. Commented Visual Basic .NET source code asset

In Figure 1 the structure of .NET comments is shown. In the particular case of method comments an extra advantage is obtained. The text is the one shown when debugging code in Visual Studio, so most programmers tend to describe widely the semantics of the functionality. Then it may be reduced the effort required to integrate the tool in the development process.

The architecture of the tool is divided in three main modules. Code Incorporation Module is in charge of uploading source code files, transforming them into documents and including documents in the repository. Documents Retrieval Module is in charge of executing queries over the corpus through IR techniques. Finally, Code Retrieval Module is in charge of processing the results obtained and get the source code related to each document and provide it to the user. In figure 2 a schema of the architecture is shown. User may upload source code assets to Code Incorporation Module. This module stores the code in the repository used by Document Retrieval Module, which sends the results of queries to Code Retrieval Module. Code Retrieval Module returns the code to the user.

In this process, as we mentioned, it is essential that comments available do not contain arbitrary semantics. Getting high quality comments depends on understanding the purpose of comments. Ambler [Ambler, 2000] establishes that the goal of source code documentation (i.e. comments) is to provide a clear idea on code functionality and design. Hence, a comment is of a better quality the better it eases code comprehension.

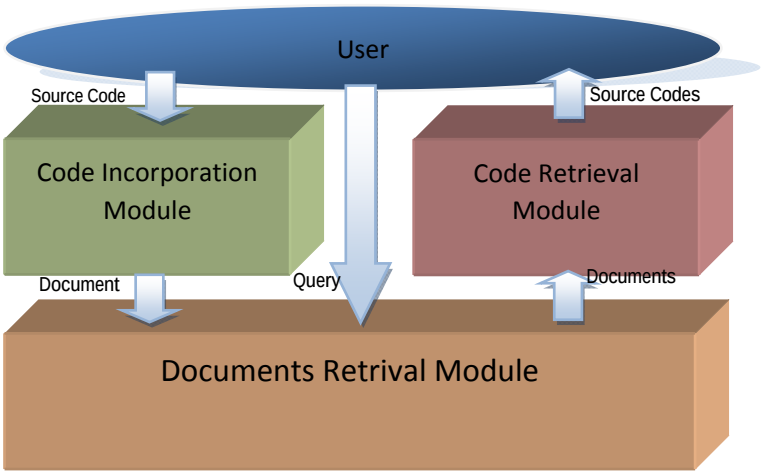

Figure 2. Basic architecture

Main aspects to consider when deciding if a comment is a quality one or not are:

- Comment contents depend on comment type
- The proper granularity level of comments is not related to comment length, but on the precision. Comments should include description of technology, functionality and design criteria.
- How comments are written. The better comments are expressed the better the quality of the comments. Incorrect or ambiguous expressions in comments may be worst than writing no comments, as long as they may affect to further maintenance.

Expressing correctly the comments is, then, crucial for further comprehension. Gutiérrez [Gutiérrez, 2007] provides some suggestions on how to write leaflets. These suggestions are of interest, as long as comments are close to leaflets in the sense of providing critical information in a concise manner. This means that quality comments are easy to read and understand, so they meet some requirements:

- They are written using simple terms
- Phrases are as short and clear as possible
- Phrases are written using active form, instead of passive form. Concatenation of sentences, pronouns, and similar linguistic resources are avoided. This is particularly important when writing in Spanish, because it is widely believed that long complex phrases indicate higher knowledge of the language.

Hence, correct writing implies ordered phrases subjected to the basic subject-verb-predicate form.

However some considerations are to be kept in mind regarding the difference from a leaflet to source code. Technical documentation shall be non ambiguous, while leaflets may not comply it because of their nature. Ambiguity is a marketing technique to attract customers asking for additional information. Attracting programmers asking about comment semantics is not the marketing we look for in Software Reuse. Only a meaning should be obtained from a given comment, this is the goal that leads to avoid the following elements from a language:

- Conditional forms of verbs. No speculation may arise, the subject of the description already exists and its behaviour is deterministic.

- Ambiguous terms, such as “further”, “some”, open lists, etc. These are the terms that we use to avoid while writing technical documents (e.g. software requirements specifications). We shall keep in mind that source code comments are a part of technical documentation.

TxReadability [University of Texas, 2007] evaluates the legibility of a given text in Spanish, English or Japanese. The feasibility on using this tool was previously shown by Gutiérrez [Gutiérrez, 2007]. Tools such as FLAVER [Santana, 1997] are capable to detect automatically the verbal form used in a phrase.

Finally it is required to decide which techniques are to be used by Documents Retrieval Module. To support expected features it is needed to provide a corpus containing grammatically categorized terms from Spanish languages [Sebastián-Gallés, 2000], using those tags from EAGLES [Monachini, 1996].

The Stopwords technique is to be used in order to optimize search. This reduces the size of the term index. An empty word does not provide useful information when selecting a document from the corpus. These techniques are implemented using EAGLES tags. Anyway we are to store the full text in order to provide exact matching capabilities.

In order to avoid restrictions derived from exact matching we decided to implement stemming. Stemming identifies the root of each term to search [Brants, 2004]. Roots are used to keep in mind, during search process, terms similar from those on the search criteria. The reason to avoid using directly the roots as search criteria is to distinguish two terms sharing the same root. This allows ranking on upper positions documents matching exactly search criteria.

Vectorial model [La Serna, 2004] has been selected to rank search results. Applying this technique requires correlation between terms and documents. On the one hand, we store the terms in a document, as well as the number of occurrences of the term. On the other hand, we need to store also the position of each occurrence. It is also required to keep in mind that those terms written in different forms, or belonging to different grammatical categories, are processed as different terms.

Conclusions

The tool described in this work intends to reach two goals. First goal consists of improving the quality of the comments within the source code. Second goal is to ease the introduction of the reuse culture in organizations, focussing on developers. In some cases programmers are who reject proper software reuse, while they receive the benefits of source code reuse.

Increasing comment quality relies on analyzing the characteristics that a good comment shall comply. The analysis we conducted allowed providing to programmers, at Technical University of Madrid, guidance on how to document the source code.

The proposal of constructing a tool meeting exposed requirements allows creating a repository containing source code. The purpose of the repository is properly reusing code safely, and locating source code through searching for functionality instead for specific and exact terms.

Further research, while prototypes are under development, is to solve the problems derived from supporting a wider spectrum of programming languages.

Bibliography

Ambler, S.W.: Writing Robust Java Code, White paper, 2000. Available at Internet < http://www.ambysoft.com/downloads/javaCodingStandards.pdf> [ref. march, 29th 2008]

Apache Software Foundation: Lucene library. Available at Internet <http://lucene.apache.org/> [ref. march, 29th 2008]

Brants, T.: Natural Language Processing in Information Retrieval. Proceedings of CLIN, 2004, pp. 1–13.

Damiani, E.: A descriptor-based approach to OO code reuse, IEEE Computer Society, 30, 1997

Google, Inc.: Google Desktop. Available at Internet <http://desktop.google.com/> [ref. march, 29th 2008]

Gutiérrez, U., Blanco, A., Casal, B., Calvo, A. y Ramos, A.: Information: new times, new media outlets, new staff (in Spanish). In Proceedings XII Jornadas Nacionales de Información y Documentación en Ciencias de la Salud. Zaragoza (Spain), 24, 2007, p. 26.

Koders: Open Source Code Search Engine. Available at Internet <www.koders.com> [ref. march, 29th 2008]

Krugle, Inc: Code search for developers. Available at Internet <www.krugle.org> [ref. march, 29th 2008]

La Serna, N., Román, U., Osorio, N., Benito, O., Espezúa, J. y Vega, H.: Estudio y Evaluación de los Sistemas de Recuperación de Información, RISI, 1(1), 2004, pp. 49-58.

Monachini, M. y Calzolari, N.: Synopsis and comparison of morphosyntactic phenomena encoded in lexicons and corpora a common proposal and applications to European languages, 1996

Prieto-Díaz, R.: Implementing faceted classification for software reuse, Association for Computing Machinery, Inc., 34, 1991

The University of Texas Accessibility Institute, 2007 June 29, 2007-last update, TxReadability. Available at Internet: <http://www.lib.utexas.edu:8080/TxReadability/app> [ref. march, 29th 2008]

TT Ying, A., L. Wright, J. y Abrams, S.: Source code that talks: an exploration of Eclipse task comments and their implication to repository mining. MSR '05: Proceedings of the 2005 international workshop on Mining software repositories. 2005, pp. 1-5.

Santana, O., Pérez, J., Hernández, Z., Carreras, F. y Rodríguez, G.: FLAVER: Automatic stemmer and inflectioner of verbal forms (in Spanish). Current Spanish Linguistic, 19(2), 1997, pp. 229-282.

Sebastián-Gallés, N.: LEXESP: Computerized Spanish lexicon (in Spanish). Edicions de la Universitat de Barcelona, Barcelona (España), 2000

Sindhgatta, R.: Using an information retrieval system to retrieve source code samples, ACM New York, NY, USA, 2006, pp. 905-908.

Authors' Information

Rubén Álvarez-González – Researcher. Technical University of Madrid. E.U. Informática. Ctra. de Valencia Km. 7. E28031 Madrid. Spain. e-mail: ruben.alvarez.gonzalez@gmail.com

Sonia Sanchez-Cuadrado – Carlos III University of Madrid. Avda. de la Universidad 30, E-28911 Leganés, Madrid, Spain. e-mail: sonia.sanchez.cuadrado@uc3m.es

Héctor García – Adjunct Professor. Technical University of Madrid. E.U. Informática. Ctra. de Valencia Km. 7. E28031 Madrid. Spain. e-mail: hgarcia@eui.upm.es