JIRiSS - an Eclipse plug-in for Source Code Exploration

Denys Poshyvanyk, Andrian Marcus*, Yubo Dong
Department of Computer Science
Wayne State University
Detroit Michigan 48202
313 577 5408
{denys, amarcus, yubo}@wayne.edu

Abstract

JIRiSS (Information Retrieval based Software Search for Java) is a software exploration tool that uses an indexing engine based on an information retrieval method. JIRiSS is implemented as a plug-in for Eclipse and it allows the user to search Java source code for the implementation of concepts formulated as natural language queries. The results of the query are presented as a ranked list of software methods or classes, ordered by the similarity to the user query. In addition to that, JIRiSS includes other advanced features like automatically generated software vocabulary, advanced query formulation options including spell-checking as well as fragment-based search.

1. Introduction

Source code searching and browsing are two of the most common activities undertaken by developers during the evolution of existing large software. These activities directly support tasks such as concept location in source code, impact analysis, change propagation, and program comprehension in general. Hence, automated tools that directly support such activities are necessary to the software developers.

JIRiSS is a software exploration tool, developed as a plug-in for Eclipse. JIRiSS supports searching in Java projects (i.e., source code and documentation), developed in Eclipse. The search is based on the indexing of the source code and associated documentation with an information retrieval (IR) method. The tool is an evolution of one of our earlier tools, IRiSS [10], which had similar functionality, but it was targeted to C++ code, as it was developed as an add-on to MS Visual Studio.

2. IR based software search - IRiSS

One of the key features of IRiSS is its information retrieval (IR) based search capabilities. In a nut-shell, IRiSS enables a programmer to search the source code of software projects using queries written in natural language. In terms of returned results, the user is given the option to choose the granularity of classes or methods.

The search engine used by IRiSS is somewhat similar to what some web search engines employ; it is based on our implementation of Latent Semantic Indexing [6]. In addition, IRiSS uses a set of tools, that transform the software into a corpus [9], which is then indexed by the IR method. Informally, the search methodology based on IRiSS consists of the following steps:

1. The software system is decomposed into text documents, creating a corpus. Comments and identifiers are extracted from the source code, as well as structural information. The user has an option to choose the desired granularity (e.g., class or method level) for documents.
2. A vector space is built using this corpus. Each document (class or method) is projected onto this space that we call the semantic search space. Steps 1 and 2 are performed once; subsequent searches start at step 3.
3. The user formulates a query, which can be in natural language. Multiple words can be used. A separate document for this user-formulated query is created and mapped onto semantic search space. The user can run any number of queries.
4. A similarity measure is computed between each document in the semantic search space and the projected user-formulated query. IRiSS returns a set of documents, ranked by the similarity measure according to the user's query.
5. The user can refine and rerun queries, based on the returned results.
6. Documents related to a specific entry from the list of results (i.e., method or class) can be also retrieved.

*Andrian Marcus is visiting in the Department of Mathematics and Computer Science at Babeş-Bolyai University, Cluj-Napoca, Romania

More details on the methodology is available in [9].

The major contribution of IRiSS is the underlying IR based technology, used during the identification of software system components most relevant to the query provided by the user. Since the results returned by the tool are ranked (see step 4 of the search methodology), this helps the user during concept location to make a decision about the concept under investigation, about the correctness and effectiveness of the formulated query, and about the overall searching procedure, after considering only a few high ranked documents.

Overall, with the experience gained while developing IRiSS, and based on encouraging results, we decided to implement the version of this tool for Eclipse, which we call JIRiSS, with additional features that should make the source code exploration tool more powerful.

3. Extending IRiSS to JIRiSS

JIRiSS keeps the major functionality of IRiSS. In addition, it has a new set of useful features and improved usability over its predecessor. Among the new features present in JIRiSS we implemented: fragment-based search, software vocabulary extraction and presentation, spell checking of the user query, word suggestions to improve queries, advanced options to customize the indexing process and the scope of the search process, etc. All these are described in the following subsections.

3.1. Fragment-based search

Solving software maintenance tasks consists primarily of three activities [5]: forming a working set of task-relevant fragments; navigating the dependencies within this working set; and repairing or creating the necessary code. Thus, we decided to provide JIRiSS with the feature to support fragment-based search of task-relevant source code fragments. This feature is very simple to use: the programmer needs to select any fragments of the source code relevant to a task and use the context menu to run it as a query in JIRiSS (see Figure 1).

JIRiSS still supports the classic way of writing queries so that users can type one query using the main dialog of the plug-in. Regardless of the way the query was generated, as soon as the query issued to JIRiSS, it computes similarities with all the indexed documents and presents the ranked results in the standard Eclipse view, named "IRiSS view" (Figure 2).

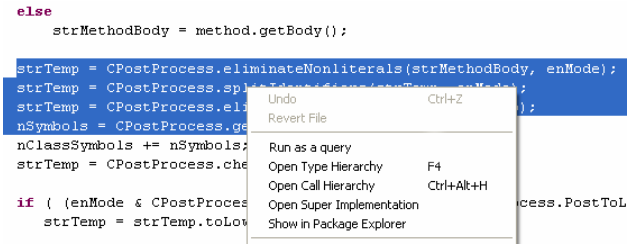

Figure 1. JIRiSS allows fragment-based search. The user selects the fragment of the source code to be used as a query and executes it with the "Run as a query" command from the context menu.

In addition, we extended the scope of indexing. By scope of indexing we mean that the user can opt to index only the source code separately, the comments separately, or the source code and the comments together.

Depending on her needs, the user can choose the appropriate indexing and searching scope. Indexing source code and comments together is the default option of JIRiSS.

3.2. Using the software vocabulary

The available technology is not the only necessary ingredient for a fast and accurate search. Most important is the query. A good search engine will return irrelevant results if inadequate query is written. One of the issues with IR based search engines is that users will often include in the query words that do not exist in the corpus. To help the user write better queries, JIRiSS generates the vocabulary of the software system and the frequency of occurrences of every word in the vocabulary (see Figure 3). This feature is especially useful for users unfamiliar with the system and its naming conventions.

Before building the vocabulary, JIRiSS does some changes on identifiers (i.e., splitting identifiers), based on observed naming conventions.

Problems | Javadoc | IRiSS View

Class	Method	Similarity
Camera	setScreenParams	0.813055
ScaleObjectTool	mouseDragged	0.807305
Camera	setScreenParamsParalle	0.80328
Camera	setSize	0.788526
ViewerCanvas	setScale	0.723998
GLCanvasDrawer	prepareView3D	0.705305
ViewerCanvas	scaleChanged	0.700639
JitterModule	setZScale	0.690434
JitterModule	setXScale	0.690117
ImageModule	setXScale	0.690104

Figure 2. Format of the ranked results returned by JIRiSS: "class | method | similarity". The user can click on any row and navigate to the source code.

Another important application of the software vocabulary that JIRiSS constructs is the automatic spell checking of the queries that user formulates. JIRiSS will not just ignore the terms that are not present in the software vocabulary will suggest the closest terms to the user for replacing the missing word.

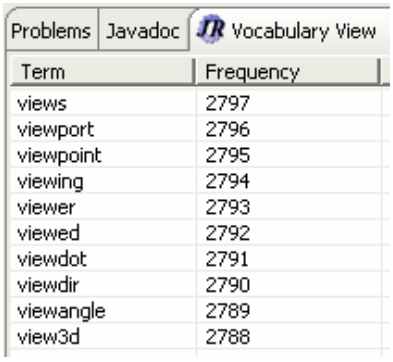

Figure 3. Part of a software vocabulary generated by JIRiSS. In this view, the user chose to view only the terms containing the characters "view".

Another mechanism for query improvement is based on term similarities that can also be obtained with IR method implemented in JIRiSS. The user just needs to indicate the subset of query terms she is interested in and JIRiSS will indicate the closest terms to those from the search space.

4. Related tools

There are many tools to support software searching and browsing using different underlying technologies. Some of these tools are Eclipse plug-ins, whereas some of these are plug-ins to other IDEs or standalone applications. Some of the best known are: sgrep [2], AspectBrowser [3] FEAT [11], SNiFF+ [13], NavTracks [12], JQuery [4], and JRipples [1].

The semantic grep (sgrep) [2] extends lexical pattern matching by light weight relational queries. The tool allows two types of queries: syntactic and semantic. Sgrep translates queries in a mixed algebraic and lexical language into a combination of grok queries and grep commands.

FEAT [11], an Eclipse plug-in, captures knowledge about the implementation of a concerns in source code. Moreover, FEAT supports locating, describing, and analysing the code implementing concerns in Java.

AspectBrowser [3] allows users to visualize programs using the map metaphor by searching for regular expressions and displaying the results graphically. It has features that help navigating through search results and handling a potentially large sets of regular expressions. Originally developed for Fortran and C, it has an Eclipse plug-in version today.

SNiFF+ [13] is a corporate "heavy weight" for browsing and searching for semantic information in source code. It has a set of integrated tools like smart searching, class browsing, symbol cross-referencing etc.

JRipples [1] is an Eclipse plug-in that supports different stages of incremental change and provides programmers with the means for program investigation and comprehension. In JRipples, developers can browse the software using the program dependence graph.

NavTracks [12] is another Eclipse plug-in that also supports browsing through the software by keeping track of the navigation history of software developers, forming associations between related files, which are used for recommendation of potentially related files.

The JQuery [4] tool supports activities involving working with cross-cutting concerns by providing better support for exploring the source code. The tool provides various statistics about cross-cutting concerns as well as presents the exploration paths during browsing activities.

Also we combined IRiSS [10] with visualization front-end, namely sv3D (source viewer 3D) [8] to support IR-based concept location with visualization [14]. Visualization is used to present alternative views of relevant search results showing their locations and distributions within the context of the complete software system. Also we visualize user's browsing history to present additional cross-query views of the results.

Lethbridge and Anquetil [7] provide an overview of earlier related tools and technologies.

5. Current and future work

As new changes are made to the software system, the semantic space needs to be reconstructed. Currently this process is initiated manually by the user. Re-indexing is done on the whole project, even though a small part of source code or comments is changed. We are working on incorporating new techniques for incremental re-indexing of semantic space.

The parsing to create the corpus is kept rather simple for scalability and efficiency purposes. While efficient, it may produce slightly inaccurate results in some extreme cases. JIRiSS is built, similarly to IRiSS, so that the user can choose to use other, more powerful parsers.

We also plan to add a feature to JIRiSS which will cluster the results of a search, based on the semantic similarity measures and/or program dependencies. With all these features in mind, we intend to integrate our tool with JRipples [1] to support various stages of incremental change.

Finally, we plan to perform a set of user studies in order to compare JIRiSS with the search features in Eclipse.

6. Demonstration of JIRiSS

To demonstrate JIRiSS, we will perform several searches during the demo. An ad-hoc comparison of the search results between JIRiSS and the Eclipse search will be done. Advantages and disadvantages of using JIRiSS will be presented. Searches on several projects will be done, to see the differences in performance depending on software sizes. The results of source code exploration will be examined, discussed and compared. Explanation on how the tool generated the results will be given.

The demonstration will be done in an interactive fashion where the process of query formulation and code exploration will require participation of the audience. People in the audience will be able to suggest queries that will be executed on the spot and the results will be discussed.

7. Availability

JIRiSS is available free of charge upon request. Please contact the authors for the most recent version of the tool.

8. Acknowledgments

This work was supported in part by grants from the National Science Foundation (CCF-0438970) and NIH (NHGRI 1R01HG003491).

9. References

[1] Buckner, J., Buchta, J., Petrenko, M., and Rajlich, V., "JRipples: A Tool for Program Comprehension during Incremental Change", in Proceedings of 13th IEEE International Workshop on Program Comprehension (IWPC'05), May 15-16 2005, pp. 149-152.

[2] Bull, R. I., Trevors, A., Malton, A. J., and Godfrey, M. W., "Semantic grep: regular expressions + relational abstraction", in Proceedings of Ninth Working Conference on Reverse Engineering (WCRE'02), Richmond, VA, October 29 - November 1 2002, pp. 267-276.

[3] Griswold, W. G., Yuan, J. J., and Kato, Y., "Exploiting the Map Metaphor in a Tool for Software Evolution", in Proceedings of 23rd IEEE International Conference on Software Engineering (ICSE'01), Toronto, Ontario, May 12-19 2001, pp. 265-274.

[4] Janzen, D. and Volder, K., "Navigating and querying code without getting lost", in Proceedings of 2nd International Conference on Aspect-Oriented Software Development (AOSD'03), 2003, pp. 178-187.

[5] Ko, A. J., Aung, H. H., and Myers, B. A., "Eliciting design requirements for maintenance-oriented ides: a detailed study of corrective and perfective maintenance tasks", in Proceedings of 27th IEEE/ACM International Conference on Software Engineering (ICSE'05), May 15-21 2005, pp. 126-135.

[6] Landauer, T. K., Foltz, P. W., and Laham, D., "An Introduction to Latent Semantic Analysis", *Discourse Processes*, vol. 25, no. 2&3, 1998, pp. 259-284.

[7] Lethbridge, T. C. and Nicholas, A., "Architecture of a Source Code Exploration Tool: A Software Engineering Case Study." Department of Computer Science, University of Ottawa, Ottawa Technical Report TR-97-07, 1997.

[8] Marcus, A., Feng, L., and Maletic, J. I., "3D Representations for Software Visualization", in Proceedings of 1st ACM Symposium on Software Visualization (SoftVis'03), San Diego, CA, June 11-13 2003, pp. 27-36.

[9] Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J., "An Information Retrieval Approach to Concept Location in Source Code", in Proceedings of 11th IEEE Working Conference on Reverse Engineering (WCRE'04), Delft, The Netherlands, November 9-12 2004, pp. 214-223.

[10] Poshyvanyk, D., Marcus, A., Dong, Y., and Sergeyev, A., "IRiSS - A Source Code Exploration Tool", in Industrial and Tool Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'05), Budapest, Hungary, September 25-30 2005, pp. 69-72.

[11] Robillard, M. P. and Murphy, G. C., "FEAT a tool for locating, describing, and analyzing concerns in source code", in Proceedings of 25th International Conference on Software Engineering (ICSE03), Portland, OR, May 3-10 2003, pp. 822-823.

[12] Singer, J., Elves, R., and Storey, M.-A., "NavTracks: Supporting Navigation in Software", in Proceedings of 13th International Workshop on Program Comprehension (IWPC'05), May 15-16 2005, pp. 173-175.

[13] SnNiFF+, "SNiFF+", Windriver, Web page, Date Accessed: 11/01/2001, http://www.windriver.com/products/html/sniff.html, 2001.

[14] Xie, X., Poshyvanyk, D., and Marcus, A., "3D Visualization for Concept Location in Source Code", in Proceedings of 28th IEEE/ACM International Conference on Software Engineering (ICSE'06), Shanghai, China, May 20-28 2006, pp. to appear.