

IRiSS – A Source Code Exploration Tool

Denys Poshyvanyk, Andrian Marcus, Yubo Dong, Andrey Sergeyev

Department of Computer Science

Wayne State University

Detroit Michigan 48202

313 577 5408

{denys, amarcus, yubo, andrey}@wayne.edu

Abstract

IRiSS (Information Retrieval based Software Search) is a software exploration tool that uses an indexing engine based on an information retrieval method. IRiSS is implemented as an add-in to the Visual Studio .NET development environment and it allows the user to search a C++ project for the implementation of concepts formulated as natural language queries. The results of the query are presented as ranked list of software methods or classes, ordered by the similarity to the user query.

A second component of IRiSS provides another searching method based on regular expression matching. This method is based on the existing “find” feature from the Visual Studio environment and it has an improved format for the display of the search results.

1. Introduction

During software development and evolution most activities involve changes to the existing source code. To carry out such tasks, software engineers spend a lot of time identifying the places where changes are to be made. Source code *searching* and *browsing* are two of the most common activities undertaken by developers [5], especially during maintenance of existing large software. These activities directly support tasks such as concept location in source code, impact analysis, change propagation, and comprehension in general.

Programmers use any knowledge available to achieve good search results and perform effective browsing. If the programmer knows and fully understands the software, these tasks become very easy and simple. In reality however, people get switched from one project to another; people performing the coding are not necessarily experts in the domain in which the software will be used and as a result may use questionable naming conventions and document code

improperly. Moreover, usually in large projects there is no single person who has knowledge of the entire system. Tool support is needed to help in these frequently performed development and maintenance activities.

2. Tools for source code searching and browsing

Three types of tools define the state-of-the-art in the field: searching tools based on the Unix shell program `grep`, such as regular expression search tools integrated in editors or development environments (IDEs), browsing facilities in IDEs based on static dependencies, and more general program understanding tools that are based on a database of facts about the software, extracted through analysis and represented in some specific format that allows for queries to be formulated and executed.

Regular expression search tools that are integrated in editors or IDEs, return an unranked list of locations in the source code where the query terms occur, the results have no semantic information attached (e.g., function name, class name, etc.) and the relationships among results are not represented. On the other hand, they are the most popular searching tools, easy to use, and quite efficient.

The browsing facilities within IDEs are based on the static analysis of the source code and help programmers quickly locate software components of interest, usually based on class and method/function names. The Smalltalk browsers are among the oldest and best of such tools. For other languages, IDEs often fall short in flexibility and lack integration with the searching features.

Finally, comprehension tools, while very powerful, are often inefficient, complicated to use, and rarely integrated with IDEs. Although today’s IDEs are much more flexible and provide support for plug-in

technology (e.g., Visual Studio .NET, Eclipse), these tools are still not widely adopted.

To overcome some of these shortcomings, we introduce a new code exploration tool called IRiSS (Information Retrieval based Software Search). IRiSS is developed as an add-in into Microsoft Visual Studio .NET. It is a research prototype, developed to support concept location in source code. It has two distinct components: one that uses an information retrieval (IR) based search engine instead of the traditional pattern-matching technique, and the other one that extends the existing regular expression search by adding to the search results the class and method names, where each line of code in the results is located.

3. IRiSS components

While designing IRiSS we considered several functional requirements [5]. In particular: advanced search capabilities and capabilities to display all relevant attributes of the items retrieved. IRiSS also addresses the following non-functional requirements: ability to handle large amount of code, respond to most queries without perceptible delay, process source code in a variety of programming languages, interoperate with other software engineering tools, integrate facilities that software engineers already use and present the user with complete information.

The popularity of the existing search feature in Visual Studio .NET and the replace feature motivated the first component of the tool. IR based search engines define the state-of-the-art in web and library searches. They offer capabilities such as formulation of natural language queries and return ranked results. All these features motivated and are implemented in IRiSS.

3.1. Regular expression based searching

The first component of IRiSS allows the user to search the source code much like the *find* feature in Visual Studio. The difference comes in the presentations of the search results. With IRiSS, the results not only show the line of code and the file, but also show the corresponding class and method and/or any other structural unit relevant to the programming paradigm. The user can sort the results of the search based on this information. Using this feature of IRiSS does not require any additional steps except having the IRiSS add-on installed.

The major change here over the *find* feature is the presentation of the search results in a new format and at a new granularity level. With the regular *find* feature, the user gets the line of text and file where the searched pattern occurs. In IRiSS the user can get the method or

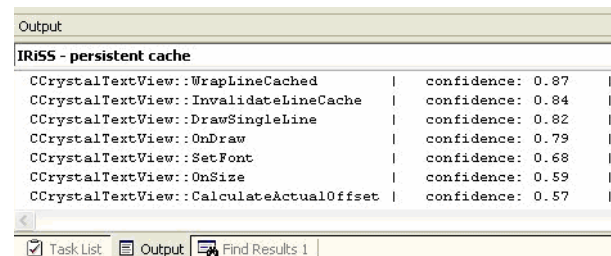
class that has most occurrences, in addition to the line of text and file.

3.2. IR based searching

The most important feature of IRiSS is its IR based search engine. It allows users to search opened projects by writing natural language queries. It returns a ranked list of related software documents (see Figure 1). The user can define the granularity of a returned document (e.g., class, method, etc).

In order to use IRiSS, a text corpus, based on the source code, is created from the open project. This is a one time step, which allows for any number of fast subsequent searches on the entire software system.

The user has an option of exploring the results with a different level of granularity: class or method. For example, if the user chooses class level granularity, the tool will compute the similarity between the user query and all the classes in the selected search space and it will produce a list of classes ranked by their relevance to the query. The results are returned in the standard .NET output view (see Figure 1) and are formatted in a meaningful manner to the software developer: namespace, class name, method name (if method granularity was selected), degree of confidence, and file name. The user can navigate to the definition or the implementation of the class or method by simply clicking on the provided links in the result list.



IRiSS - persistent cache	
CCrystalTextView::WrapLineCached	confidence: 0.87
CCrystalTextView::InvalidateLineCache	confidence: 0.84
CCrystalTextView::DrawSingleLine	confidence: 0.82
CCrystalTextView::OnDraw	confidence: 0.79
CCrystalTextView::SetFont	confidence: 0.68
CCrystalTextView::OnSize	confidence: 0.59
CCrystalTextView::CalculateActualOffset	confidence: 0.57

Figure 1. The format of the ranked results returned by IRiSS: "class::method name" | "confidence"

The search engine used by IRiSS is somewhat similar to what some web search tools employ; it is based on our implementation of Latent Semantic Indexing [4]. In addition, IRiSS uses a set of tools that transform the software into a corpus [6], which is then indexed by the IR method. Figure 2 shows how IRiSS works and how it is used. 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 the 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.

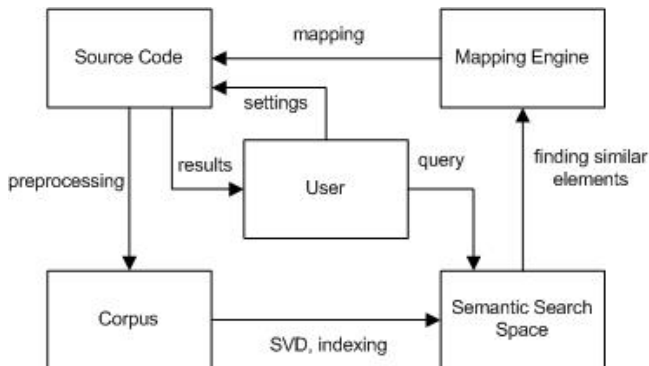


Figure 2. The searching process using IRiSS

The major contribution of this source code exploration tool 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.

3.3. Additional features

IRiSS also has a set of additional features that can help developers to fine tune the corpus creation. Using these features, developers can customize the corpus being indexed to include/exclude numerical values, preserve the original identifier names, split identifiers, index *.h files. The user can also use an external vocabulary for “stop words”, that is words that will not be included into the corpus. The corpus can be exported to XML or plain text so it could be used by other external IR engines.

4. Related tools

There are several tools to support software searching and browsing.

The semantic grep (sgrep) [1] 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 [7], an Eclipse plug-in, captures knowledge about the implementation of a concern 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+ [8] 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.

Zhao et al. introduce the tool to support static non-interactive approach to feature location by using information retrieval (IR) to compute similarities between features in source code along with static representation of the source code using branch-reserving call graph [9].

Lethbridge and Anquetil [5] provide an overview of earlier related tools and technologies.

5. Current and future work

As changes to the software system occur, the semantic space needs to be reconstructed. Currently, the user initiates this process of re-indexing. We are working on an automated re-indexing method that will account for user changes when the user is in idle mode.

When the user uses words in the query that do not exist in the software, they are ignored by IRiSS. In the future, IRiSS will spell check the query with respect to the vocabulary of the software and suggest changes.

Advanced search features will also be added to IRiSS such as return results without specific words. The .NET find feature and IRiSS will be integrated.

The parsing to create the corpus is kept simple for scalability and efficiency reasons. While efficient, it may produce slightly inaccurate results in some extreme cases. IRiSS is built such that the user can choose to use other, more powerful parsers. In fact, we also have a version that uses Columbus [2] to parse the source code, which allows flexible customization of initial corpus.

IRiSS is implemented in C++ as an add-in component to MS Visual Studio .NET. While the underlying IR technology of IRiSS is essentially language independent (natural and programming), it currently works only with solutions implemented in C/C++. The corpus creation part is the one that depends on the programming language. We plan to add parsers for C#, Java, and VB so that we could work with solutions that contain C# and VB source code as well. We also implemented a beta-version of the parser for Java source code. We plan to release a version of IRiSS as Eclipse plug-in as well.

Finally, we also plan to add a feature to IRiSS, which will cluster the results of a search based on the semantic similarity measure or program dependencies.

6. Demonstration of IRiSS

To demonstrate the features and the technology of IRiSS we will perform several searches during the demo. Two software tools will be used to perform source code exploration tasks: IRiSS and the standard "Find in Files" Visual Studio feature. This will allow easy comparison of the results.

Source code exploration will be done on two open-source projects and several queries will be tested: WinMerge (<http://sourceforge.net/projects/winmerge/>) and Doxygen (<http://www.stack.nl/~dimitri/doxygen/>).

The differences in the processes for each tool (IRiSS and "Find in Files") will be emphasized and discussed during the demo. Advantages and disadvantages of each tool will be presented. The results of source code exploration will be examined, discussed and compared. Explanation on how the tool generated the results will be given.

The demo will be held 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

IRiSS is available free of charge upon request. Please contact the authors for a copy.

8. Acknowledgements

This work was supported in part by a grant from the National Science Foundation (CCF-0438970).

9. References

- [1] 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.
- [2] Ferenc, R., Siket, I., and Gyimóthy, T., "Extracting Facts from Open Source Software", in Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04), Chicago, IL, September 11 - 14 2004, pp. 60-69.
- [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, Toronto, May 12, 2001, pp. 265-274.
- [4] 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.
- [5] 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, Technical Report TR-97-07, 1997.
- [6] Marcus, A., Sergeev, 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 (WCRE2004), Delft, The Netherlands, November 9-12, 2004, pp. 214-223.
- [7] 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, Portland, OR, May 3-10, 2003, pp. 822-823.
- [8] SnNiFF+, "SNiFF+", Windriver, Web page, Date Accessed: 11/01/2004, <http://www.windriver.com/products/html/sniff.html>, 2001.
- [9] Zhao, W., Zhang, L., Liu, Y., Sun, J., and Yang, F., "SNIAFL: towards a static non-interactive approach to feature location", in Proceedings of 26th International Conference on Software Engineering (ICSE'04), Edinburgh, Scotland, May 2004, pp. 293-303.