Source Code Exploration with Google

Denys Poshyvanyk, Maksym Petrenko, Andrian Marcus, Xinrong Xie, Dapeng Liu

Department of Computer Science
Wayne State University
Detroit, Michigan USA 48202
{denys, max, amarcus, xxr, dliu}@wayne.edu

Abstract

The paper presents a new approach to source code exploration, which is the result of integrating the Google Desktop Search (GDS) engine into the Eclipse development environment. The resulting search engine, named Google Eclipse Search (GES), provides improved searching in Eclipse software projects.

The paper advocates for a component-based approach that allows us to develop strong tools, which support various maintenance tasks, by leveraging the strengths of existing frameworks and components. The development effort for such tools is reduced, while customization and flexibility, to fully support user needs, is maintained.

GES allows developers to search software projects in a manner similar to searching the internet or their own desktops. The proposed approach takes advantages of the power of GDS for quick and accurate searching and of Eclipse's extensibility. The paper discusses usage scenarios, advantages, limitations, and possible extensions of the proposed tandem.

1. Introduction

During software evolution most activities require the users to understand large and often new parts of a software system. Most of the time, the developers rely on the source code and the available documentation alone to achieve this understanding. In this context, source code searching and browsing are two of the most common activities undertaken by developers [4]. These activities directly support such tasks as concept location in source code, impact analysis, change propagation, debugging, and comprehension of software in general. Given their widespread use, we need to support them with fast and accurate tools and techniques.

Traditionally, most developers were (and still are) using *grep* (global regular expression print) for regular expression search, or variants of it, to help them locate items of interest in the source code and documentation. Existing integrated development environments (IDE) feature powerful searching tools also based on regular expression matching and on other mechanisms. For example, in addition to its native search features, Eclipse is currently extended with a number of research tools that support source code searching and exploration [1, 3, 8, 10, 11].

While each of these tools is geared to support searching in the context of a specific development or maintenance task, they have yet to make it in the mainstream of the software development practice with Eclipse. As with most research prototypes, they suffer from development problems and limited interaction with potential users, thus delaying their wide adoption in industry. Despite some of its problems and limitations, *grep*-based approaches for source code searching are very popular as they rely on simple to use and robust technology, which is familiar to and understood by most developers.

To overcome some of these problems, we propose in this paper a new approach to source code searching, which combines an existing off-the-shelf component for searching, namely the Google Desktop Search (GDS, http://desktop.google.com), with the Eclipse development environment. The resulting combination, named Google Eclipse Search (GES), leverages the strengths of GDS, the extensibility of Eclipse, and of course their popularity, familiarity, and trust among the developers.

2. Motivation for the proposed approach

With the advent of the Internet in the past decade, searching for information in various formats has been redefined by the internet search engines, most of them being based on information retrieval (IR) indexing techniques. IR-based searching, which usually allows formulation of queries with multiple words, today is more popular as regular expression matching used to be.

We took advantage of this popularity and simplicity of use and proposed a software searching methodology and technology based on IR [6]. In order to bring the technology to the fingertips of developers, we also integrated our toolset with MS Visual Studio [9] and Eclipse [8]. Our efforts still suffer from some of the same problems most research prototypes do, especially in terms

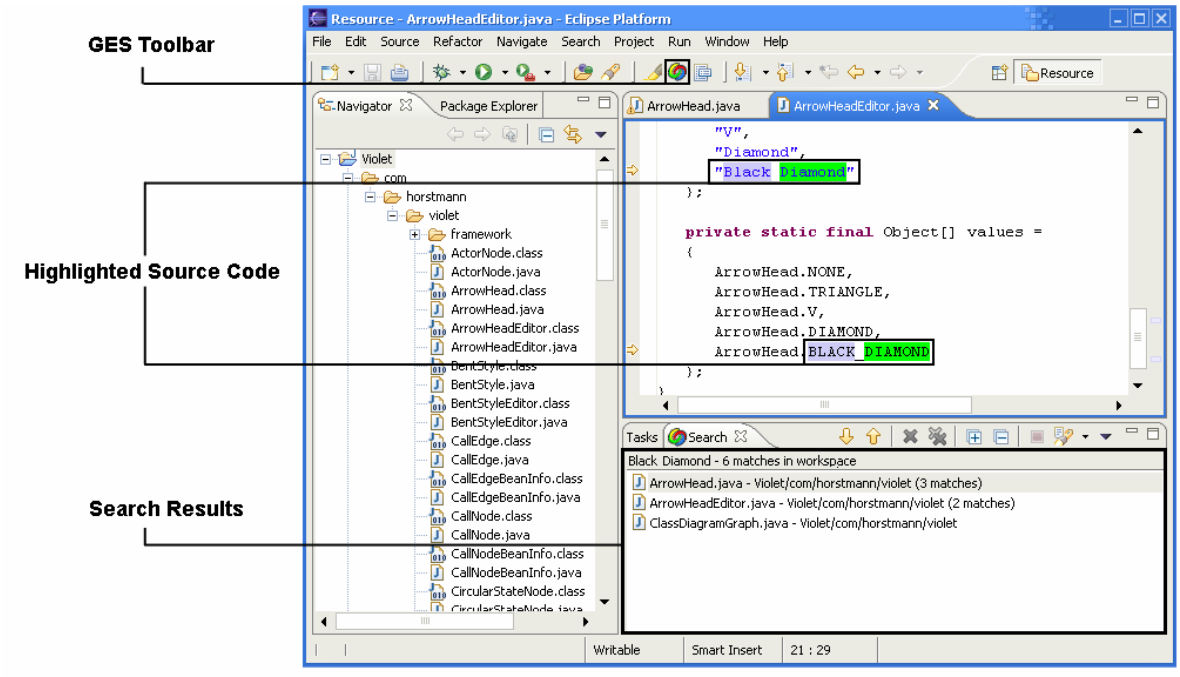

Figure 1. The GES plug-in for Eclipse showing the results for the query "black diamond"

of computational efficiency and the online re-indexing of the software as it changes during evolution.

Still, we strongly believe in and support the IR-based approach to software searching, especially given the positive results of comparison [5] and combination with other approaches [7]. To bring the technology closer to industry adoption, we need to solve the existing problems, highlighted above.

Recently, Google scaled its technologies to desktop level by releasing Google Desktop Search (GDS) [2]. One of the aspects that set GDS apart from some of its competitors is the possibility to write extensions for GDS using the Google Desktop SDK (http://desktop.google.com/developer.html). These extensions allow Google Desktop to index various types of applications and files. We are specifically interested in using GDS to index and search source code files and associated project files.

GDS has most of the features that were missing in our previous efforts, namely efficiency and the ability to unobtrusively re-index the search space as it changes.

GDS could be used to search the files of a software project as it is, through the use of an Internet browser. However, such a use would be awkward as it would break the work flow of the developers as they would have to constantly switch between the IDE and the browser.

We decided to extend the standard searching capabilities of the Eclipse environment with the searching power provided by GDS. In addition to what we mentioned before, GDS has several features that make it an attractive choice for this integration with Eclipse to support searching in software projects: on-the-fly preprocessing and indexing of the context; continual indexing, which maintains and updates content location changes for more accurate results; immediate response to user queries; history of searches; advanced search options based on Boolean operators; sorting of the results by relevance or date; etc.

The integration between GDS and Eclipse has advantages over other software search solutions:

- features specific to IR-based searching, such as multiple term queries, natural language queries, Boolean operators, and ranking of search results;
- scalability and high reliability of the proven search engine (i.e., GDS), which is important for massive file repositories, such as large scale software systems;
- display of and access to the search results within Eclipse's IDE, using its native interfaces that provide direct links between the search results and the actual source code in the editor.

3. The GES design and implementation

GES is implemented as a plug-in for the Eclipse development environment (see Figure 1) to be used for searching within projects or within a custom working set of files using natural language (or multiple word) queries.

The GES search dialog is displayed in the standard Eclipse search dialog panel and the search results are presented through the standard search results presentation view (see Figure 1).

The GES search experience is similar to Eclipse's *File Search*. In order to perform a search using GES, the user has to type a query into the GES search dialog and specify the scope of the search (i.e., workspace, selected resources, enclosing projects, or working set). After the execution of the query, the search results are displayed within the GES search results tab (see Figure 1). The results can be easily explored by simply browsing the files in the editor. When source code files are in the scope of the search, the terms from the original query that are found in the java file are highlighted with colors (see Figure 1). Note that the terms from the query need not be in immediate vicinity of each other in the source code.

Through GES, the user can take advantage of all the intrinsic features of GDS, including searching using a set of terms, exact phrases, queries with Boolean operators, or restricting the search results to specific file types (i.e., by using the "filetype:" modifier).

In order to communicate with GDS from Eclipse, we employ the GDS Java API (http://desktop.google.com/plugins/javaapi.html), which can be used to access GDS from any java application.

Also, since GDS is not an open-source application, the only possible way to customize it currently is through the available GDS SDK. This issue raises several challenges in building and using GES.

One of the major issues is the GDS' background indexing. By default, GDS indexes (and re-indexes) the user files only when the user's computer is idle; thus, to be able to initially use it, the user typically needs to wait until GDS completes the (re-)indexing of the files. Unfortunately, currently this problem can not be addressed using GDS preferences or GDS API calls. Ideally, we would like to allow the user the option to choose when and how the files to be (re-)indexed.

4. Using GES for concept location

In this paper, we describe a pilot case study to show how GES can be used for concept location and source code exploration. The study is performed on an open-source software system Violet (http://www.horstmann.com/violet/). Violet is a cross-platform UML editor written in Java. Violet version 0.15, used for this study has 65 classes with 448 methods with about 9,000 lines of source code.

For the demonstration purposes, we choose the following software engineering scenario, which typically requires a lot of source code searches – the request for a new feature. Thus, we formulated the following change request: "*introduce a user-defined arrow type for the class diagram*". In other words, we tried to find the location in the source code for the implementation of the concept that requires adding an arrow of a new type to be used in UML class diagrams.

The location methodology followed in this example is based on [5].

4.1. Queries and results

After reading the change request, the basic search terms were extracted and used for the initial query in the attempt to identify the parts of the source code where Violet deals with arrows in class diagrams: "*arrow class diagram"*. Unfortunately, the first query did not return any matching files.

After a brief inspection of Violet's GUI, we discovered that the arrow end is called *head* and the arrow line is called *edge*. Thus, the next search query was formulated according to this new information. The next query, "*edge class diagram*", returned 11 files in the following order: *UseCaseDiagramGraph, StateDiagramGraph, SequenceDiagramGraph, StateTransitionEdge, ObjectDiagramGraph, NoteNode, ObjectNode, FieldNode, ImplicitParameterNode, ClassDiagramGraph,* and *CallNode*.

Examination of these files showed that the *ClassDiagramGraph* class was relevant to the concept. To verify this finding, the required change was implemented and resulted in modifying the "*draw*" and "*getPath*" methods in the *ArrowHead* file, adding new variables in the *ArrowHead* and *ClassDiagram* files, and modifying existing variables in the *ArrowHeadEditor* and *ClassDiagram* files. In addition, the settings file *ClassDiagramGraphStrings.properties* was modified to support newly created type of arrow.

4.2. Comparing GES with the file search

In order to compare GES with the standard Eclipse search feature, we asked another programmer, who was not familiar with the process and the results of concept location with GES, to perform a couple of searches using standard *File Search* feature in Eclipse, including the one presented above.

Using GES turned out to be helpful for queries that do not take into account the format of identifiers in the source code. For example, for another concept location task, where we tried to locate the place in the source code which specifies the width of the class diagrams, the File Search programmer needed to perform three searches: "*Default Width*", just "*Width*", and search for "*Default*" within the results of the "*Width*" query. This 3-step chain was caused by the fact that the default width value in Violet was stored in *DEFAULT_WIDTH* variable – that is why the search for "*Default Width*" ended up with no results. On the other hand, using GES the user was able to take advantage of text preprocessing capabilities of Google and obtained the results with the very first "*default width*" query. Of course, the same results can be

obtained through the File Search by using the regular expression "*Default.*Width*". Nonetheless, to construct such expressions, the programmer should have additional information about identifiers in the source code that contain these two terms; plus, it may be just unusable to construct such complex expressions all the time to search for simple identifiers.

Next, although the concept has been located by the two programmers using both GES and the File Search, the main observation after analysis of the collected data showed that the developer using GES investigated less lines of code in order to locate this particular concept. Partially, as it was mentioned before, this was caused by the fact that the File Search programmer had to perform the searches within the query results to narrow down these results, while the GES programmer was able to specify all the available search terms in the first query. Also, since GES returned a ranked list of results, the developer had the potential to learn relevant information faster than with File Search.

As this study had a proof-of-concept role, we do not generalize these conclusions. Further, more detailed case studies are needed to properly extend the results.

In order to gain more insight into the scalability of GES, we ran several queries on bigger projects than Violet and compared the performance with that of the standard Eclipse *File Search*. The queries were executed on P4 2.8Ghz with 1GB of RAM. We ran the GES plug-in and the *File Search* in Eclipse 3.

The first software for this study was Art of Illusion (AOI), a 3D modeling studio that supports rendering and animation. AOI is written in Java and it has 442 classes, 20 interfaces, and 100,838 LOC. The second software for the study was Eclipse version 3.1 with the complete sources. It included approximately 20,000 files totaling approximately 2 millions LOC.

Table 1. Average response time per query for the GES and the Eclipse File Search (in full seconds)

	Violet 9 KLOC	AOI 100 KLOC	Eclipse 2 MLOC
GES	1 s	1 s	1 s
File Search	1 s	9 s	97 s

Ten queries were run on each system and the average response times needed for GES and the *File Search* is shown in Table 1. The results show that GES is more effective in terms of response time than the *File Search*. The results also show that GES scales up very well with the size of the search space. However, it should be mentioned that it takes some time for GDS to initially index the files for large projects. As this is a one-time step, it only affects the first search on a software system.

5. Conclusions

Integrating GDS into Eclipse is a solution to improve source code searching and produce an easier to adopt approach to this problem. GES allows Eclipse software developers to perform searches in the source code and associated documentation of a software system, using most features offered by GDS. Early data indicates that GES is considerably faster than the Eclipse *File Search* and users find it easy to use.

6. Acknowledgements

This research was supported in part by grants from the National Science Foundation (CCF-0438970) and the National Institute for Health (NHGRI 1R01HG003491).

7. References

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

[2] Cole, B., "Search engines tackle the desktop", in *IEEE Computer*, vol. 38, 2005, pp. 14-17.

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

[4] Lethbridge, T. C. and Nicholas, A., "Architecture of a Source Code Exploration Tool: A Software Engineering Case Study", Univ. of Ottawa, Ottawa Tech. Report TR-97-07, 1997.

[5] Marcus, A., Rajlich, V., Buchta, J., Petrenko, M., and Sergeyev, A., "Static Techniques for Concept Location in Object-Oriented Code", in Proc. of 13th IEEE Int. Workshop on Program Comprehension (IWPC'05), 2005, pp. 33-42.

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

[7] Poshyvanyk, D., Gueheneuc, Y., Marcus, A., Antoniol, G., and Rajlich, V., "Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification", in Proceedings of 14th IEEE International Conference on Program Comprehension (ICPC'06), Athens, Greece, 2006, pp. 137-148.

[8] Poshyvanyk, D., Marcus, A., and Dong, Y., "JIRiSS - an Eclipse plug-in for Source Code Exploration", in Proc. of 14th IEEE International Conference on Program Comprehension (ICPC'06), Athens, Greece, June 14-17 2006, pp. 252-255.

[9] Poshyvanyk, D., Marcus, A., Dong, Y., and Sergeyev, A., "IRiSS - A Source Code Exploration Tool", in Industrial and Tool Proc. of 21st IEEE International Conference on Software Maintenance, Budapest, Hungary, Sept. 25-30 2005, pp. 69-72.

[10] Shonle, M., Neddenriep, J., and Griswold, W., "AspectBrowser for Eclipse: a case-study in plug-in retargeting", in Proceedings of OOPSLA workshop on eclipse technology eXchange, 2004, pp. 78-82.

[11] Singer, J., Elves, R., and Storey, M.-A., "NavTracks: Supporting Navigation in Software", in Proc. of Int. Workshop on Program Comprehension, May 15 2005, pp. 173-175.