Support for Static Concept Location with sv3D

Xinrong Xie, Denys Poshyvanyk, Andrian Marcus
Department of Computer Science
Wayne State University
Detroit Michigan 48202
{xxr, denys, amarcus}@wayne.edu

Abstract

The paper presents a new visualization approach to support static concept location in source code. The approach is realized through the combination of two existing tools: IRiSS, which is an information retrieval based tool that support source code searching and browsing; and sv3D, which is a software visualization front end. Both tools are integrated into MS Visual Studio .NET.

The motivation behind the approach, the definition of the visual mappings, and usage examples are also presented in the paper, together with an outline of future and related work.

1. Introduction

Searching and browsing the source code is one of the most common activities in software engineering as it directly supports concept location, a key step during the analysis and comprehension of software. Concept location techniques are mainly divided into static and dynamic; most common static techniques [8] are based on string matching, dependency search, and information retrieval. There is little effort done in trying to support these techniques with visualization. Most existing efforts focus on the visualization of dependency graphs. One of the difficulties in visualizing the results of a string matching search, for example, is the granularity and size of the results of a given search.

This paper presents the combination of two software comprehension tools to provide visualization support for information retrieval (IR) and string matching based concept location. One tool (i.e., IRiSS – Information Retrieval based Software Search [11]) implements an IR based concept location method and combines it with the existing regular expression based search in MS Visual Studio .NET. The other tool (i.e., sv3D – source viewer 3D [7]) is a visualization front end for software data.

2. Related work

Various visualization tools were developed to support source code browsing through different representations (e.g., FEAT [13], Aspect Browser [3], SHriMP [14], etc). Some visualization tools use graphs to represent the relationship between software artifacts. In these graphs, nodes may represent the system entities and edges may represent relations between the entities. For example, Rigi [15] is used to represent in a graph-like visualization, application component relationships and dependencies. SHriMP [14] shows inheritance relationships in an software system through the use of nested graphs using the fish-eye filtering technique. CodeCrawler [5] displays classes and the relationships between them using polymetric views, where the size and the color of nodes are used to represent software metrics.

Some of the above visualization tools provide searching functions. For example, SHriMP supports three searching strategies: general, artifact, and relation search. Others are integrated with existing IDEs (i.e., CodeCrawler in Smaltalk and SHriMP in Eclipse [6]).

3. Program comprehension using sv3D

Source Viewer 3D (sv3D) [7] is a software visualization framework that uses a 3D visual metaphor based on the Seesoft [1, 2] file and pixel maps. In particular sv3D creates 3D renderings of the software data based on user defined mappings. It uses poly cylinders to represent software entities (e.g., characters, lines of code, methods, functions, classes, etc.) and it uses containers to show aggregates of these elements (see Figure 3). The height and color of the poly cylinders can be used to represent various software metrics or other data relevant to the software system. sv3D is designed to interact easily with analysis tools, as it has no analysis component. It supports advanced user interactions and usage of the

3D space for visualization, such as: the *overview* feature that can show large amounts of source code in one view; *zooming* and *panning* at variable speeds; two types of 3D manipulators (i.e., *track ball* and *handle box*) are available to the users to interact with the visualization; a number of *filtering* methods, such as *transparency*; the user can take snapshots of the current view, thus a sequence of the snapshots can be used to keep a *history* of the visualization.

Sv3D can be used to support tasks such as: fault localization, visualization of execution traces, source code browsing, impact analysis, evolution, slicing, etc.

4. Concept location with IRiSS

Concept (or feature) location is one of the most frequent activities that software developers perform when there is a need to change something in a software system. It is routinely done during incremental change of software [12]. An overview of different methods is available in [16] and [8]. In this paper, we will concentrate only on the IR-based concept location method [9]. We developed a tool [11] (i.e., IRiSS) that implements the IR-based concept location technique as a plug-in to MS Visual Studio .NET.

In a nutshell, the concept location process using IRiSS has the following steps:

1. Preprocessing the source code and documentation;
2. Building a vector space representation of a software system using Latent Semantic Indexing (LSI) [4];
3. Executing queries formulated by the user;
4. Retrieving and analyzing the results, which are returned as a ranked list of source code elements.

Steps 1 and 2 are done usually once, while steps 3 and 4 are performed repeatedly by the user during concept location. IRiSS uses the Visual Studio interface for the presentation of the search results in text format (method or class prototypes).

5. Motivation for merging sv3D and IRiSS

There are several reasons to merge sv3D with IRiSS. The IRiSS interface, as most searching tools and the Visual Studio *find* feature, is text-based. One problem with such an interface is that it does not provide the user with an overview of the results of a search and the relationships between the source code elements. Visualization can solve this problem by providing a visual map of the source code elements related to a query and the relationship between each other. Their distribution in the system is important and may be lost in the textual representation. Filtering the results, presented in a visual format, is also more natural for users.

In addition, during concept location, the user needs to keep in mind more than the results of the most recent query. It is important, for example, to see if a certain method or class was returned as a result to several previous queries in this process. Textual representations may present difficulty in showing this type of information. Once again, visualization would help in representing such information.

As methods and classes in the software system are investigated, it is often more useful to have additional information about these classes than just the source code. sv3D can display this information together with the result of a search.

Finally, integrating sv3D with one or more development environments is one of our original goals. Since IRiSS is integrated in the Visual Studio .NET, it provides a good communication mechanism between the IDE and sv3D.

In the typical envisioned usage scenario, after IRiSS generates results for a user query, the results are transferred to sv3D for visualization based on the user preferred mapping. Then, the user can explore the results either in the IDE or in sv3D. While navigating in sv3D, the user is able to double-click on a poly cylinder, which represents a method and navigate directed to the corresponding source code in the IDE's text editor.

6. Visualization for concept location

Based on the user needs during the concept location process, the following information lends itself well for visual representation with sv3D:

1. similarities between a user query and the methods or classes in the system;
2. similarities between a given method and the other methods or classes in the system;
3. similarities between a class and other classes in the system;
4. process information such as: the number of hits on a particular method or class during successive searches; historical data that shows the sequence of classes and methods that are investigated while searching.

We can define a simple visualization in sv3D to represent this information by using mappings to the color and height of the poly cylinders. Through the use of transparency, the user can easily filter the results of interest.

6.1. Using color to represent similarities

In this early version of the tool integration, we use color in sv3D to represent the similarities between the

types of source code elements mentioned earlier, based on a pre-defined (or user defined) color scheme. The similarity measure between two source code documents, or a query and a source document in IRiSS is defined as the cosine between their corresponding vectors in the LSI space [9], with a [-1, 1] range. The pre-defined mapping for color used to visualize the similarities is described in Table 1.

Table 1. Color scheme for similarity representation

Similarity range	Color
0.5 – 1	Red
0.4 – 0.5	Orange
0.2 – 0.4	Light green
0 – 0.2	Dark green
-1 – 0	Grey

Poly cylinders can be mapped to methods or classes in the source code, and the color of the poly cylinder will show the similarity value to a given user query (see Figure 1). Users can define their own preferred mapping if they choose so (e.g., use green or yellow for high similarity values), with a different granularity and/or number of colors. Intensity could also be used instead of hue.

Figure 1. Using colors to represent similarities. Each poly cylinder represents a method here.

6.2. Using height to represent the browsing history

The power of sv3D is using the third dimension to represent data through the height of poly cylinders. In this application, we use the height of the poly cylinders to represent information about the history of the browsing in a sequence of related searches, during concept location. For example, the height of a poly cylinder corresponding to a method is mapped to the number of visits by the user to that method, during the search (see Figure 2).

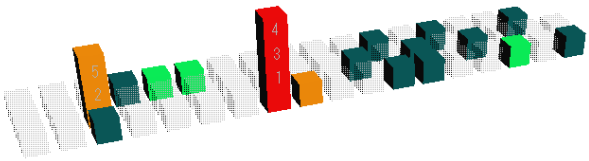

Figure 2. Using the height of the poly cylinders to represent the number of visits. Each poly cylinder represents a method here. Transparency may be used to focus on relevant information.

Height can also be used to show the sequence of the explorations as in Figure 3. Every time the user explores the method by clicking the corresponding poly cylinder, the height of the poly cylinder will increase by one unit and the number of steps in the exploration sequence will be shown on the poly cylinder. We consider this information as a useful addition for source code browsing and exploration since programmers can easily find out how many times they have explored each method and they can see the map of their exploration process.

While using sv3D it is possible to focus on the specific parts of the results by filtering the output using transparency. For example, this may be useful when the user wants to visualize similarities only in a specific range or keep methods with similarities that have been visited at least once (see Figure 2).

7. Example of using IRiSS and sv3D for concept location

To show how sv3D can be used in conjunction with IRiSS, we describe an example. Concept location is performed on an open-source software system WinMerge 2.0.2 (www.sourceforge.net/projects/winmerge). The system had been chosen because we have previous experience and knowledge about the source code.

WinMerge is a Win32 tool for visual difference display and merging, for both files and directories. One of the most important features of WinMerge is the side-by-side line differencing and visualization. The system supports a unicode and a flexible syntax coloring editor. More advanced features include windows shell integration and regular expression filtering.

The WinMerge version we used here consists of 69 classes with 624 methods. The system size totals in about 64,000 lines of source code (comments and lines of code, excluding blank lines).

In order to show how IRiSS and sv3D can be effectively combined for concept location we formulated the following change request: *"Implement a new feature to swap the context of the currently opened text files in the WinMerge system"*.

The feature that needs to be located to carry out the change in this case is the implementation of displaying two separate files in windows within the WinMerge application. The concept most likely includes the classes that *wrap* the textual representation of files and classes that link files to the particular textual representation on the screen.

The following subsections describe the process step-by-step, with certain details omitted and show how sv3D will display relevant information during concept location.

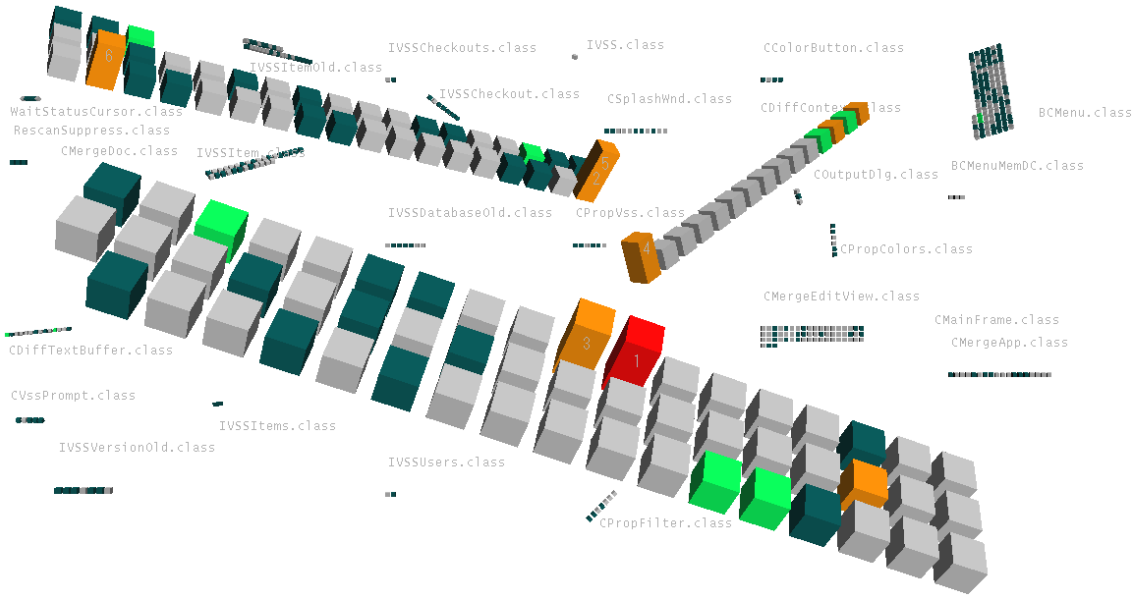

Figure 3. The results of the first query run on WinMerge. Each poly cylinder represents a method and containers represent classes. Color is mapped to the similarity measure based on the default mapping defined in Table 1.

7.1. Query formulation

The graphical user interface of WinMerge is based on the Multiple Document Interface (MDI) of the MFC framework. Taking into account properties of GUI applications [10] that have the structure predefined by the GUI framework and text messages describing the functionality of the GUI elements (that can be successfully used during the concept/feature location), we formulated the following initial query: "*left right file*". We extracted these terms during analysis of the menu items in WinMerge application since we thought they are relevant to the concept related to the change request.

IRiSS was configured using default settings. Description of those setting can be found in [11]. By default, the granularity used by IRiSS to build the LSI space is at method level. In other words, at each search, IRiSS will return as results a list of methods from the system under investigation.

IRiSS checks first the user queries to see if the words composing it are present in the source code. Every term in this query was present in the dictionary of the software system, also built by IRiSS. We decided to start the concept location with this query because the change request primarily deals with opened files in the left and right hand side panels.

The query was submitted to IRiSS, which returned a list ranked methods in descending order, based on the similarity measure to the query. As it was mentioned earlier, for the purpose of this example, we did not use IRiSS's interface to browse the results, but sv3D to represent them. Figure 3 shows the results of this first query. Information about any poly cylinder may be obtained by a double click. The red poly cylinder in the figure corresponds to the DoFileOpen method in the CMainFrame class and it has a 0.500104 similarity measure with the first query. It is the most similar method to the query and it was easy to spot. Of course, it would have been just as easy to get this method in the textual representation, where it would be on the top of the list. Where the visualization pays off at this stage is in assessing which class has most methods that are similar to the query. Such information is helpful in deciding what part of the source code will be investigated next.

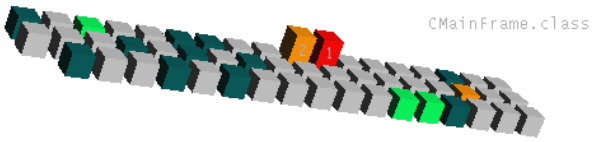

Figure 4. Examining the DoFileOpen and ShowMergeDoc methods from the CMainFrame class

7.2. Browsing similar methods

Using sv3D, we identified the following methods that appeared most similar to the original query: CMainFrame::DoFileOpen (see Figure 3, Figure 4, and Figure 5) and CMainFrame::ShowMergeDoc (see Figure 3, Figure 4, and Figure 5) and CDiffContext::UpdateTimes (see Figure 6).

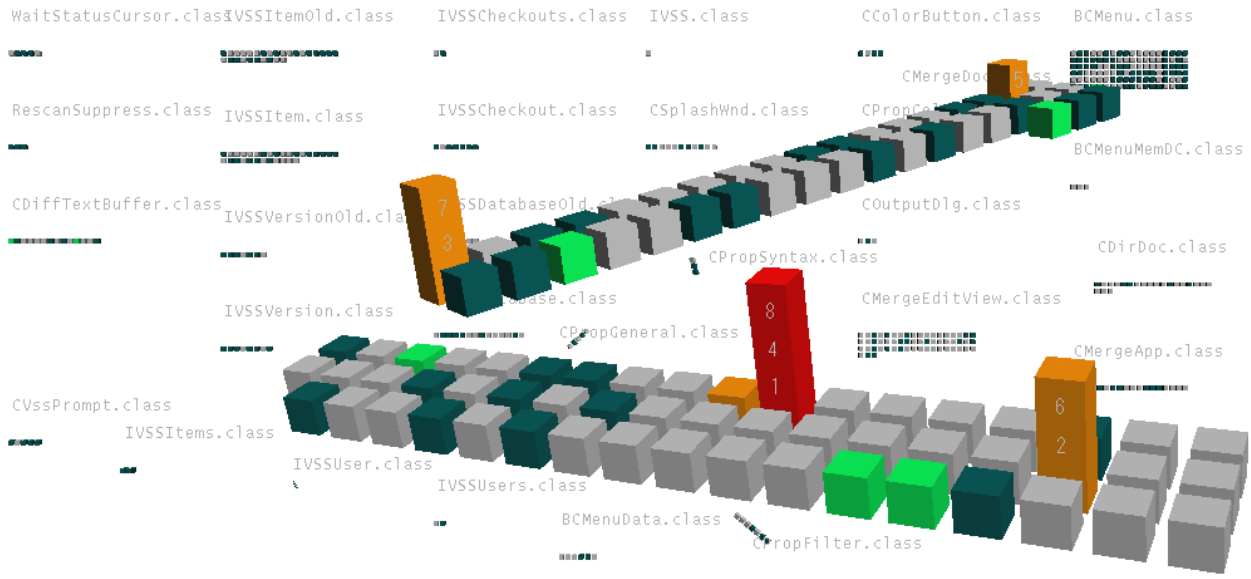

Figure 5. Representation of the WinMerge system with a focus on the classes that have browsing history information represented using the height of the poly cylinders. Color is mapped to the similarity measure to the query. Each poly cylinders shows and method and each container represent a class. The numbers on the poly cylinders indicate the visitation order of the corresponding methods.

By clicking on the respective poly cylinders the focus is transferred to the text editor in Visual Studio.

Analysis of these methods revealed that only CMainFrame::DoFileOpen was the part of the concept. However, it does not represent the whole concept since it delegates this process to other objects which are to be identified during the concept location.

Figure 6. Examining the UpdateTimes method from the CDiffContext class.

7.3. Query refinement

Taking into account the knowledge obtained during examination of these three methods, the query was refined as described in [9]. The resulting query was: *"file left right files open merge edit selected top"*.

As a result of running this query, a set of other methods was investigated. In this round, we investigated the CMergeDoc::SetMergeViews (see Figure 7) and the CMainFrame::OnDropFiles methods.

The CMergeDoc::SetMergeViews method revealed the final details about the implementation of the concept. This method sets particular views for the files being displayed and the class CMergeDoc, which wraps all the necessary "document/view" details needed to store and display a document on the screen.

7.4. Using the browsing history to identify relevant documents

In this example we skipped some of the details about visiting other methods using sv3D. For complex concepts, multiple queries may need to be run and several methods to be investigated. In this process, sv3D can help represent the browsing history. In other words, whenever the user requests to compute pair similarities for every method in the system with a selected method, the browsing activities in the search space of related methods may be recorded by counting the clicks on the respective methods. This number may be represented by the height of the poly cylinders (see Figure 5 and Figure 8).

Figure 7. Examining the SetMergeViews method in CMergeDoc class.

This representation allows the identification of "interesting spots" during the process of concept location. At every moment of time, the user is able to consider additional information to the similarity measures. All the methods visited during this location process will stand out. This means that those methods were similar to several of the previous queries and deemed important at different stages of the process.

For example in Figure 5 the method with the highest similarity (colored with red) has been visited 3 times during the concept location. Visiting sequences (numbers on the bars) can also be recorded and visualized and are shown in the figures.

Figure 8. Examining the history of visits along with similarities for CMainFrame

8. Conclusions and future work

Enriching source code searching and browsing with visualization support will improve concept location and other similar activities. The experience of merging two technologies and tools, IRiSS and sv3D, to achieve this showed promising results.

Recording and visualizing historical data of such searching activities may also prove to be useful to the software engineers.

As future work, we plan to experiment with different mappings and also to include dependency and other information in the visualization to improve the location process. Evaluations are needed and planned as well.

The same representation can also be used to show clusters of software elements, which would support other comprehension and analysis activities.

Finally, IRiSS and sv3D need to be more tightly coupled, such that one could use sv3D to select a method in the system and use it as a query in IRiSS.

9. Acknowledgement

We are grateful to Louis Feng, Jonathan Maletic, Andrey Sergeyev and Yubo Dong, who contributed to the development of sv3D and IRiSS.

10. References

[1] Ball, T. and Eick, S., "Software Visualization in the Large", Computer, vol. 29, no. 4, April 1996, pp. 33-43.

[2] Eick, S. G., Steffen, J. L., and Sumner, E. E., "Seesoft - A Tool For Visualizing Line Oriented Software Statistics", IEEE Transactions on Software Engineering, vol. 18, no. 11, 1992, pp. 957-968.

[3] Griswold, W. G., Yuan, J. J., and Kato, Y., "Exploiting the Map Metaphor in a Tool for Software Evolution", in Proceedings of 23rd International Conference on Software Engineering (ICSE'01), Toronto, Ontario, May 12-19 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] Lanza, M. and Ducasse, S., "Polymetric Views—A Lightweight Visual Approach to Reverse Engineering", IEEE Transactions On Software Engineering, vol. 29, no. 9, September 2003, pp. 782-795.

[6] Lintern, R., Michaud, J., Storey, M. A., and Wu, X., "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse", in Proceedings of 1st ACM Symposium on Software Visualization, June 11-13 2003, pp. 47-56.

[7] 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.

[8] Marcus, A., Rajlich, V., Buchta, J., Petrenko, M., and Sergeyev, A., "Static Techniques for Concept Location in Object-Oriented Code", in Proceedings of International Workshop on Program Comprehension, 2005, pp. 33-42.

[9] Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J., "An Information Retrieval Approach to Concpet 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.

[10] Michail, A., "Browsing and searching source code of applications written using a GUI framework", in Proceedings of 24th International Conference on Software Engineering, May 19-25 2002, pp. 327-337.

[11] Poshyvanyk, D., Marcus, A., Dong, Y., and Sergeyev, A., "IRiSS - A Source Code Exploration Tool", in Proceedings of IEEE International Conference on Software Maintenance, September 25-30 2005, pp. to appear.

[12] Rajlich, V., "A Methodology for Incremental Change", in Extreme Programming Perspectives, Marchesi, M., Succi, G., Wells, D., and Williams, L., Eds., Reading, MA Addison Wesley, 2002, pp. 201-214.

[13] 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.

[14] Storey, M.-A. D., Best, C., and Michaud, J., "SHriMP Views: An Interactive Environment for Exploring Java Programs", in Proceedings of Ninth International Workshop on Program Comprehension (IWPC'01), Toronto, Ontario, Canada, May 12-13 2001, pp. 111-112.

[15] Storey, M.-A. D., Wong, K., and Müller, H. A., "Rigi: a visualization environment for reverse engineering", in Proceedings of IEEE International Conference on Software Engineering (ICSE'97), Boston, MA, May 17 - 23 1997, pp. 606-607.

[16] Wilde, N., Buckellew, M., Page, H., Rajlich, V., and Pounds, L., "A Comparison of Methods for Locating Features in Legacy Software", Journal of Systems and Software, vol. 65, no. 2, 15 February 2003, pp. 105-114.