Interaction and Visualization Techniques for Programming

Mikkel Rønne Jakobsen

Dept. of Computing, University of Copenhagen
Copenhagen, Denmark
mikkelrj@diku.dk

Abstract. Programmers spend much of their time investigating the source code of a program, which often involves navigating and understanding delocalized code fragments. This Ph.D. project explores the use of information visualizations that are designed to support programmers in these activities. I use controlled experiments to provide precise measurements of the usability of visualizations and detailed insight into users' interaction with visualizations. Also, case studies are used to understand how professional programmers use visualizations in realistic work activity. Overall, this research will contribute empirically founded insight into the design and use of visualizations in programming.

Application

My Ph.D. thesis has the working title "Interaction and Visualization Techniques for Programming". My scholarship is running from June 1st 2006 until May 31st 2009. This paper describes the research problems that form the basis of the thesis and outlines the dissertation research. I have also submitted a short paper to Interact 2007 called "Transient Visualizations".

My advisor is associate professor Kasper Hornbæk (kash@diku.dk), Department of Computing, University of Copenhagen.

The research area for this project is human computer interaction and information visualization. The research topic is empirical studies of the use of information visualizations to support programming.

Introduction

Programming is a complex human activity. Programmers are often required to develop correct source code from a general description of how the program should work. Programmers spend much of their time reading and understanding existing code, which typically requires navigation between delocalized fragments of code. As the source code grows in size and complexity, navigating and understanding the source code becomes mentally demanding.

Information visualization concerns the use of interactive visual representations of data to support problem solving. Use of information visualization to support programming has received considerable attention from the research community [e.g., 9,12]. Relatively few studies, however, have examined the effects of using visualizations in programming. Further empirical studies are needed to understand better how users interact with tools in modern programming environments and how visualization techniques can enhance these interactions.

The complexity of activities performed by programmers that are engaged in challenging tasks has implications for both design and evaluation of visualizations. First, it is complicated to design usable visualizations that support the diversity of activities that goes on in programming. Second, it is difficult to make realistic studies that yield generalizable results. For example, source code editing is rarely considered in design and evaluation of visualization techniques, even though writing and editing source code is an integral part of programming. Thus, techniques that have a documented positive effect on program understanding may prove impractical for programming work in general. Furthermore, empirical research is often limited to studying participants performing simple tasks, often with insufficient experience with the programming tools that are used. Thus, further research is needed to found a broad and deep empirical insight into the use of visualization in programming tools.

An overall hypothesis for this Ph.D. project is that information visualizations can support navigation and understanding of source code. To investigate this claim, visualizations are designed and evaluated. An important aim of this research is to combine visualizations with tools in modern programming environments to support the complex and varied work of professional programmers.

Related work

Recent studies of professional programmers performing maintenance tasks indicate that navigating and understanding source code can be difficult and time consuming [1,10,11]. Ko et al. [10] studied 10 expert Java programmers performing maintenance tasks in a modern programming environment and found that participants spent up to 35% of their time navigating between dependencies in the code. Latoza et al. [11] reported from an observational study of 7 professional programmers that the biggest usability problem involved getting lost while navigating around the source code. Specifically, reading and understanding code fragments that are located in different parts of a source code file requires programmers to scroll or jump between locations in the file [1]. Also, the files containing code fragments related to a task may be located in different parts of the hierarchy of source code. Navigating between code fragments in source code comprised of thousands of files requires considerable efforts of the user. These findings indicate a potential for improving programming tools to better support navigation and understanding of source code.

Furnas's fisheye views [6] present an approach to supporting navigation and understanding of source code. In fisheye views, all source code lines are assigned a degree of interest based on their a priori importance and their relation to the line in focus. Lines with a degree of interest below some threshold are removed from the

view. The resulting view provides both focus details and context information. Combining a fisheye views with a source code editor may help programmers navigate and understand the source code: it may improve understanding, and reduce the need to navigate, because lines of source code that are useful given the programmer's focus of attention are shown in the view; it may allow programmers to navigate faster, because target information is shown in the fisheye view.

Intuitively, visualizations that represent hierarchical data structures by visual enclosure seem useful for representing hierarchically organized source code. Two-dimensional representations of the entire source code provide programmers with an overview that may potentially support navigation between delocalized source code. Earlier research of two-dimensional representations of source code has focused on other aspects. For example, SeeSys applies space-filling techniques to visualize statistics in code that is hierarchically divided into subsystems, directories and files [2]. Thumbnail views of source code arranged in two-dimensional space may facilitate the forming of a spatial memory of the source code to support navigation [4]. Evaluations may prove such two-dimensional arrangements of source code useful. However, scaling such arrangements of thumbnails to show large amounts of source code in its entirety may be difficult. In contrast, space-filling techniques such as Treemaps can scale to very large amounts of data [5].

Research method and contributions

This research project mainly contributes empirically founded insight into the use of visualizations in programming. My research aims to strengthen the empirical literature on information visualization. For researchers, important areas of further work are pointed out. For designers, my work may provide advice about how information visualizations can be combined with modern programming environments.

I use elements from different empirical research methods. Controlled experiments are conducted to provide precise measurements of usability of visualizations and to describe users' interaction with interfaces that contain visualizations. Case studies are used to provide evidence about the benefits of visualizations used by professional programmers doing normal work in their own environment. Combining different methods can potentially increase the validity and trustworthiness of the outcome [8].

Prior research

Fisheye views of source code

My first research contribution published at CHI 2006 reported an experimental study of a fisheye view of source code [7]. The design that was evaluated combined a fisheye view with a source code editor by dividing the editor window into two areas: the focus area comprised the editable part of the view and the context area contained mainly readable context information. Semantic relations between parts of the source code were included in the calculation of the degree of interest. The fisheye view was fully integrated with the Eclipse development environment.

The experiment compared the usability of the fisheye view with a common, linear presentation of source code. Sixteen participants performed tasks significantly faster with the fisheye view, although results varied dependent on the task type. The participants generally preferred the fisheye view and analysis of participants' interaction with the fisheye view showed effective use of the context information to complete tasks. Results indicated that semantically related information is important, while source code displayed because of a high a priori degree of interest is less useful. Limitations of the experiment call for follow-up studies. First, participants in the experiment were given relatively short practice time with the fisheye view and were thus not confident in using the fisheye view. Second, realism of the programming environment was reduced because the tools available to the participants in the experiment were limited. Third, the study investigated only simple programming activity. Questions raised include (1) what types of information to include, (2) how to effectively use display space, and (3) how to improve interaction in the fisheye view to facilitate both investigation and editing of source code.

Transient visualizations

As mentioned above, a fisheye view was found to support participants in task involving navigation and understanding. However, some participants expressed concern that the fisheye view could be inappropriate for writing and editing code because they want a large view of source code for those tasks [7]. One idea is to allow users to call up the fisheye view on demand.

In general, transient use of information visualization close to the user's focus of attention presents an approach to support specific contexts of use without permanently changing the user interface. Using transient visualizations to facilitate infrequent and unpredictable contexts of use, the original permanent view can be dedicated to information used in frequent contexts of use. For example, information needed for navigating to dependencies in the code could effectively be shown on demand in a transient fisheye view of source code. Experimental studies are planned as part of this project to investigate the usefulness of transient visualizations, in particular with regard to programming.

To obtain empirical evidence about the usefulness of transient visualizations, I conducted an experiment. Twenty participants performed search and navigation tasks using map interfaces. One interface contains an overview permanently fixed in the upper-right corner of the detail window. In another interface, users can temporary call up an overview at the mouse cursor position. Results show clear preference for the interface with the transient overview. However, no overall difference between the two interfaces was found for task completion times and accuracy.

Future research

Fisheye views of source code

Currently, I plan a controlled experiment to compare interfaces that use transient and permanent fisheye views of source code. Fig. 1 shows part of the interface that uses the transient fisheye view of source code. The experiment combines a study of

participants performing maintenance tasks, with measurement of usability in repeated trials involving navigation and understanding tasks. The experiment aims to uncover whether transient fisheye views can be used for navigating and understanding source code efficiently, without compromising on a large view of code for reading and editing. Additionally, participants will get adequate training to be proficient with the interface prior to measurements of usability.

Next, I am planning a case study to investigate how professional programmers use fisheye views in real work. The study aims to observe participants and collect quantitative information about their use of fisheye views of source code. Establishing a protocol of programming work will enable detailed analysis of interaction patterns in the use of fisheye views compared with a traditional, linear view of source code. A prerequisite for the study to yield useful results is a stable and usable interface that programmers will use in their work.

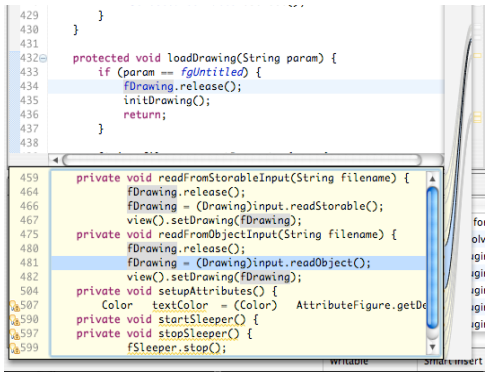

Fig. 1. Detail screenshot of a transient fisheye view of source code that uses popup windows above and below the editor window to show program lines related to the user's focus.

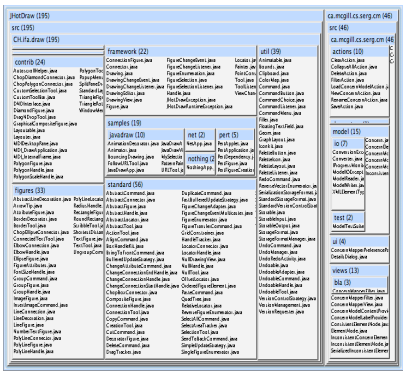

Fig. 2. Preliminary design of a map visualization of Java source code using a treemap approach.

Transient visualizations

Results from the ongoing experiments with transient visualizations that are part of this project will be consolidated to provide an empirical basis for discussing the benefits and limitations of transient visualizations.

Map visualizations of source code

Two-dimensional map visualizations are investigated with the aim of supporting understanding and navigation of source code. Fig. 2 shows a preliminary design of such a map, which represents the entire source code of a Java program. Map visualizations may utilize spatial cognition to provide programmers with an overview of the entire source code. One use of such a map is to highlight those delocalized code fragments that are being investigated in a maintenance task, for example by highlighting code that has been read or edited. Other possible uses include visualizing dependencies in the source code to support understanding or visualizing results of a search in source code to help programmers find and navigate relevant code.

An initial study is planned to uncover important factors in the design of the visual representations in maps. For example, the spatial layout of the source code representation must be stable to facilitate spatial cognition. Also, appropriate information residue or landmarks are needed to find information for navigating in the source code. User studies of participants performing tasks that target search and spatial memory are needed to investigate whether maps of source code can support the forming of spatial memory. I plan controlled experiments to compare the use of maps to conventional tools, and case studies to examine how programmers will adopt map visualizations in their programming work practices in the long term.

References

1. Brian de Alwis, Gail C. Murphy (2006). Using Visual Momentum to Explain Disorientation in the Eclipse IDE. Proc. VL/HCC, p. 51-54.
2. M. J. Baker and S. G. Eick (1995). Space-filling software visualization. Journal of Visual Languages and Computing, 6(2):119–133.
3. B. B. Bederson, B. Shneiderman, and M. Wattenberg (2002). Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. ACM Trans. Graph., 21(4):833–854.
4. R. DeLine, M. Czerwinski, B. Meyers, G. Venolia, S. Drucker, and G. Robertson (2006). Code thumbnails: Using spatial memory to navigate source code. Proc. VLHCC, p. 11–18.
5. J.-D. Fekete and C. Plaisant (2002). Interactive Information Visualization of a Million Items, Proceedings of InfoVis'02, p.117.
6. G. W. Furnas. (1981) The FISHEYE view: A new look at structured files. Technical Report #81-11221-9.
7. M. R. Jakobsen and K. Hornbæk (2006). Evaluating a fisheye view of source code. ACM CHI Conference, p. 377–386.
8. J. E. McGrath (1995) Methodology Matters: Doing Research in the Behavioural and Social Sciences, in Baecker, R. M. et al. (eds.) Readings in Human-Computer Interaction, Morgan Kaufmann Publishers Inc.
9. B. A. Price, I. S. Small and R. M. Baecker (1992). A Taxonomy of Software Visualization, Proceedings of the 25th Hawaii Int. Conf. on System Sciences, p. 597-606.
10. A. J. Ko, H. Aung, and B. A. Myers. (2005). Eliciting design requirements for maintenance-oriented ides. Proc. ICSE, p. 126–135.
11. T. D. LaToza, G. Venolia, and R. DeLine. (2006). Maintaining mental models: a study of developer work habits. Proc. ICSE, p. 492–501.
12. M.-A. Storey (2001). SHriMP views: an interactive environment for exploring multiple hierarchical views of a Java program. ICSE 2001 Workshop on Software Visualization.