A Structured Demonstration of Five Program Comprehension Tools: Lessons Learnt

Susan Elliott Sim
Dept. of Computer Science
University of Toronto
10 Kings College Rd,
Toronto ON, Canada
M5S 3G4
+1 (416) 978 4158
simsuz@cs.utoronto.ca

Margaret-Anne Storey
Dept. of Computer Science
University of Victoria
PO Box 3055 STN CSC
Victoria, BC Canada
V8W 3P6
+1 (250) 721 8796
mstorey@uvic.ca

Andreas Winter
Institute for Software Technology
University of Koblenz-Landau
D-56016 Koblenz, Germany
+49 (261) 287 2764
winter@uni-koblenz.de

Abstract

The purpose of this panel is to report on a structured demonstration for comparing program comprehension tools. Five teams of program comprehension tool designers applied their tools to a set of maintenance tasks on a common subject system. By applying a variety of reverse engineering techniques to a predefined set of tasks, the tools can be compared using a common playing field. A secondary topic of discussion will address the development of "guinea pig" systems and how to use them in a structured demonstration for evaluating software tools.

1 Background

Many tools to support program comprehension have been developed in both industry and research during the past few years. Although these tools share the common goal of simplifying the task of understanding large software systems, they have different functionality and different approaches. Various tools extract system artifacts and their relationships at different levels of granularity, ranging from fine-grained data at the abstract syntax tree level to coarse-grained data at the architectural level. Other tools support techniques for abstracting these artifacts e.g. using metrics-based discovery of software architecture, query-based investigations of software structure, or browsing capability to support a structured exploration of the system. Furthermore, tools can be distinguished by how they present the results of analyses. Some tools show different kinds of graphs, reporting the results in terms of source code, or presenting these data in textual representations such as lists or tables.

Despite the commonly held assumption that effective tools could be of huge economic gain, there have been relatively few tool evaluations. The evaluations that have been done have limited results that cannot easily be transferred to other tools or studies. There is no single way to demonstrate the facilities of the various program comprehension tools and compare tool capabilities in a realistic reengineering context in an equitable manner.

Over the past few years, a consensus has been developing within the reverse engineering community that more evaluation of tools and more methods to evaluate are needed. One approach that has sparked a great deal of interest is the benchmark technique where tools are deployed on a common subject system, or "guinea pig." In this vein, Chikofsky organized a demonstration project where participants analyzed the WELTAB III Election Tabulation System (presented at CSMR'98 in Florence).

A derivative of the benchmark technique, a structured demonstration was held at CASCON99. CASCON is an IBM sponsored Canadian software technology conference held annually in Toronto. Sim and Storey invited six teams of developers to bring their tools to participate in this workshop. In addition to the common subject system, the teams were given a set of assigned tasks. These tasks were designed from the point of view of a software engineer who was an end-user of the tools. This event occurred in "real time," that is, the six teams analyzed the subject system at the same time and in the same place using different tool sets. This structured demonstration is discussed further in a technical paper in this volume. [1]

In February of 2000, Winter organized a Workshop on Algebraic and Graph-Theoretic Approaches in Software Reengineering at the University of Koblenz-Landau. [2] At this workshop, four program comprehension tools and their underlying approaches were discussed from a more theoretical point of view. One of the conclusions from the workshop was that the participating approaches were based on similar concepts and that a comparative demonstration of the tools would be very informative. The participants agreed to apply their tools to the subject system and tasks from the CASCON99 structured demonstration.

2 WCRE 2000 Panel

A panel at WCRE 2000 was organized to provide a forum for the teams from both workshops to review their experiences with the structured demonstration materials. In total, five tool teams will report on their experiences with this benchmark: three of the original six teams from the CASCON99 workshop and an additional two teams from the Koblenz workshop. Each of the five teams submitted a short paper describing their experience. In these papers, they discuss their results, amount of time spent on the tasks, changes made to their tools to complete the tasks, other tools used to solve the tasks, and their recommendations for future changes which should be made to their tool. During the panel there will be individual team presentations as well as a group discussion to discuss the commonalities and interesting differences between the tools and approaches. The panel will close with a discussion on the benefits and costs of using a benchmark system with a common set of tasks for comparing and evaluating tools.

2.1 Participating Tools

The following tools are participating in the panel:
From the CASCON99 Workshop:

- Rigi, University of Victoria. Rigi is a tool for re-documenting and browsing software architectures.
- PBS, University of Waterloo. PBS (Portable Bookshelf) is a tool suite for extracting, analyzing, and visualizing software architecture.
- UNIX tools such as grep, emacs etc.

From the Koblenz Workshop:

- GUPRO, University of Koblenz, is a graph-based, generic environment to support program comprehension based on query technology and graph algorithms.
- Bauhaus, University of Stuttgart. Bauhaus is a set of tools for architecture recovery using static analysis.

2.2 Subject System and Tasks

The subject system was *xfig 3.2.1*, an open source drawing program consisting of approximately 60KLOC of ANSI C code. Teams were given two sets of tasks, reverse engineering and maintenance. The reverse engineering tasks asked the teams to document the structure of the system and to do an assessment of the architecture. There were three maintenance tasks, a defect removal, a feature change, and a feature addition. These tasks are described in greater detail in the developer handbook which was handled to the participants as part of their task specification. [3]

The teams that participated in the CASCON demonstration had already done the tasks and did not need to do any further work (however, they were free to do more analysis if they wished). Since GUPRO and Bauhaus participated after the CASCON99 workshop, they were requested to carefully keep track of how long they spent on parsing the code and on each of the tasks. They were also asked to document any changes made to their tools in addition to recording any other tools they used to help do the assigned tasks. It was suggested that they spend approximately one day on the assigned tasks, since this was the amount of time spent by the teams at CASCON.

3 Observations and Theses

Observations were made by the organizers and participating tools in the structured demonstration. These observations lead to a set of theses to be discussed.

3.1 Tools and Applications of Tools

3.1.1 Parsing

All participating teams (except for the UNIX team) reported problems with their parsers and had to adapt them to parse *xfig*.
Thesis: To avoid redundant work in building and adapting parsers, it would be beneficial to use a common set of robust and correct parsers.

3.1.2 Flexibility

Analyses performed with GUPRO, PBS, and Rigi are based on an adaptable conceptual model. When working on the *xfig* tasks, the teams used or adapted conceptual models developed for other reengineering problems.
Thesis: Tools supporting program comprehension and software maintenance require flexible conceptual models that can be modified as the user or task requires.

3.1.3 Quantity of Extracted Data

The quantity of data extracted from the *xfig* source by the tools varied a great deal size. The size of the database depended on the granularity of the underlying conceptual models. Fine grained conceptual models lead

to an enormous amount of extracted data, which can barely be processed by program comprehension tools.
Thesis: Careful consideration must be given to when fine-grained source code representations should be used. The benefits of such an analysis need to be commensurate with the costs.

3.1.4 Level of Experience With the Tool

The teams participating in the tool demonstration were very familiar with their tools. A thorough understanding of the tools was necessary to adapt the tool to the given problems and to solve the maintenance tasks.
Thesis: Using powerful program comprehension tools requires a thorough knowledge about the tools.

3.1.5 Participation

It was not difficult to find teams willing to participate in the structured demo. It would be interesting to know why teams wanted to participate in a *tool contest*.
Thesis: A structured demo provides a lot of insight for tool designers into their own tools and allows them to directly compare their tool capabilities with other tools and learn about future tool extensions.

3.2 Demonstration Scenario

3.2.1 Tool Selection

In addition to their own tools, all teams used other tools, such as plain UNIX tools to solve the given maintenance tasks.
Thesis: Analyzing and comparing program comprehension tools should focus on a suitable mix of different interoperable tools instead of viewing single stand alone tools.

3.2.2 Educational Value

Some teams had novice users participating in the demonstration. They found that having novices work alongside expert users enabled them to learn about the tool quickly and easily.
Thesis: A structured tool demonstration provides good educational material to support teaching novice users.

3.2.3 Fairness

Not all of the participating tools in the structured demonstration were specifically designed for supporting maintenance tasks like the one given in the *xfig* scenario. Some of the tools are more suited to more focused subtasks in program comprehension.
Thesis: Any evaluation must be situated within a particular context. For the structured demonstration, the viewpoint was that of a software maintainer. The given scenario is fair in the sense that it was applied to all the tools.

3.2.4 Replication

The application of the structured demonstration by Bauhaus and GUPRO in a second trial led to similar results to those obtained in the original demonstration. A lot of the results from the CASCON99 workshop were verified in this second run.
Thesis: Structured tool demonstrations should lead to repeatable and comparable results. The demonstration defined for CASCON99 fulfilled this requirement.

3.2.5 Scalability of Results

The *xfig* guinea pig only addresses a very small maintenance problem which could be processed in one day. Industrial strength software is usually much bigger and maintenance problems are usually much more complex.
Thesis: This small tool demonstration example was able to accentuate lots of tool features which may be transferable to large-scale maintenance problems.

4 Future Plans

We are planning two future events. Since parsing was a difficulty for so many tools, we plan to have another structured demonstration for comparing parsers. As noted above, many of the tools that participated are suitable for particular subtasks in a maintenance scenario. We are therefore planning a collaborative reengineering exercise to combine tools to help solve maintenance and reengineering tasks. We are open to suggestions for other events and for feedback on the structured demonstration described in this panel.

References

[1] S. E. Sim and M.-A. D. Storey, “A Structured Demonstration of Program Comprehension Tools,” presented at Working Conference on Reverse Engineering, Brisbane, Australia, 2000.

[2] “Workshop on Algebraic and Graph-Theoretic Approaches in Software Reengineering Available" at <http://www.uni-koblenz.de/~winter/AlGra/>.

[3] “A Collective Demonstration of Program Comprehension Tools." Available at <http://www.csr.uvic.ca/cascon99>.