On Selecting Software Visualization Tools for Program Understanding in an Industrial Context

Scott Tilley
Department of Computer Science
University of California, Riverside
stilley@cs.ucr.edu

Shihong Huang
Department of Computer Science
University of California, Riverside
shihong@cs.ucr.edu

Abstract

This paper discusses issues related to selecting software visualization tools for program understanding in an industrial context. Visualization tools are often advocated as an effective means of aiding program understanding by creating graphical representations of the subject system through reverse engineering. However, there are numerous practical considerations related to tool selection for large, real-world projects that are regularly unaddressed. For example, the choices are often limited to commercial tools that are widely available, that have a low cost of deployment and training, and that easily integrate into existing software engineering processes. To illustrate some of these unique issues, two types of constraints imposed on the process of software visualization tool selection to support program understanding in the domain of embedded, real-time control systems are described.

Keywords: software visualization, program understanding, tool selection, industry

1. Introduction

There are two broad categories of documentation that are used as an aid in program understanding [2]. The first category is textual, exemplified by printed manuals and user guides. A more flexible form of textual documentation is electronic, such as HTML or XML files, which permit activities such as automated indexing and hyperlinking between document fragments.

The second category of documentation is visual, exemplified by graphs and charts. A more flexible form of visual documentation is also electronic, which permits actions such as interactive layout and user-directed editing of the graphs. Visual documentation can be used in conjunction with textual documentation, to provide a more holistic approach to program understanding.

We have been involved in a project with an industrial partner whose primary goal is to create a methodology for the reuse of legacy software. One of the first steps towards reuse is the redocumentation of the software system to aids in its understanding. To support this goal, the project has a secondary goal of constructing a prototype toolset supporting the methodology. A key requirement of this toolset is the ability to visualize various aspects of the software system under scrutiny in several different ways. This requirement lead to extensive investigation into the capabilities of current software visualization tools, and the creation of a comparison framework for evaluating the tools' capabilities in the context of the industrial partner's numerous constraints.

The next section describes one of the most stringent restrictions on the tool selection process: corporation-wide policies that are used for all software applications. Section 3 discusses the project-specific requirements that augment the corporation-wide policies. Section 4 provides an overview of the rational for selecting a particular visualization tool that was ultimately used by the industrial partner when the toolset was deployed at their institution. Finally, Section 5 summarizes our findings.

2. Corporation-Wide Policies

There were a number of corporation-wide policies that are strictly enforced by the industrial partner's engineering division that directly influence tool selection. Three of the most important were adoption costs, hardware platform, and software integration.

2.1 Adoption Costs

An over-arching consideration is that the methodology resulting from the project must be directly applicable, usable, and supportable in the context of the real-world day-to-day needs of company's software

developers. This means the goal of toolset adoption is particularly important in this setting. Achieving this goal means guiding the methodology development and tool selection in very specific ways. For example, the methodology should be relatively easy to integrate into existing documentation and development processes already in use at the company. Failure to plan for deployment has been an Achilles heel in many reengineering-related efforts.

Another important issue concerned post-deployment support. The company reasonably requested that professional support be available for whichever tools were selected. This requirement immediately ruled out almost all academic and research tools, even if they showed the promise of superior capabilities, because by definition these are not commercially-available products.

A final consideration related to deployment and integration is toolset familiarity. It was highly desirable for the selected tools to be something that the company's developers were already familiar with. Although this requirement did not rule out new tools, it did impose a consideration of tool training and adoption cost.

2.2 Hardware Platform

A matter closely related to deployment and integration is the hardware platform. The selected tools had to run under Microsoft Windows NT 4, the operating system currently supported by the company's information technology group for the developers. This meant that several advanced tools only available on Solaris could not be considered for immediate deployment.

There is a way to make Solaris-based tools available to Windows users, either through an X-Windows interface or the use of client software such as Citrix MetaFrame [1]. However, these options were deemed to be too intrusive (violated the integration policy discussed above), therefore such tools were removed from consideration for the current project.

2.3 Software Integration

Another matter closely related to adoption cost is software integration. The final tools selected for the project had to smoothly integrate with the existing applications already in use. For the documentation tools, this meant reading and writing industry-standard formats for textual documents.

It also meant working with the configuration management system that is used to guide the build process. The documentation in the industrial partner's environment is closely linked to the software artifacts, and these links are enforced by both corporate policy and build procedures.

3. Project-Specific Requirements

In addition to the corporate policies regarding tool usage stated above, there were numerous project-specific requirements for the software visualization tools. These requirements were driven in part by the nature of the project, both in terms of the application domain of embedded real-time control systems, and the software system itself, which imposed special considerations that had to be addressed.

Many of these considerations were influenced by the overall project goal of supporting code reuse through automatic document creation using reverse engineering technologies. For example, determining and subsequently visualizing code fragments that are dependent on the operating system.

Perhaps the most stringent constraint placed on the tool selection process was due to the unique characteristics of a real-time embedded control system. For example, this means the selected tool(s) should have the ability to identify hardware-specific software components, incorporate real-time data gathering into its analysis, and trace timing dependencies from high-level tasks down to low-level bit variables. Moreover, the result of this analysis must be suitable for visual documentation.

The following is a representative list of some of the project-specific issues that influenced the software visualization tool selection:

3.1 Source Code

Since the source code of the subject system is written in a highly-optimized manner (to overcome some of the limitations of the implementation language), the engineers' questions are often related to low-level issues:

- Which memory layout is used? Is there an internal and an external flash available and which code is located in which area?
- How are the variables used and in which type – reading, writing, or both?
- What naming convention is used and in which way are the global and local variables characterized?

3.2 Software/Hardware Interface Analysis

The software under analysis is a real-time control system that runs on an embedded platform, leading to

questions regarding the mapping between software and hardware functionality:

- Which software components are directly reliant on special-purpose hardware?
- Which parts of the system are processed serially and which parts in parallel? How does the software determine this scheduling?
- Which parts of the software run on which parts of the hardware?

3.3 Runtime Behavior

One of the most difficult aspects of real-time systems is related to timing considerations:

- Which parts of the software are calculated in the time-dependent tasks and which parts are calculated in the angle-dependent sequences?
- Which functions run in which task(s)?
- Is the runtime behavior identical to the old system?

These project-specific requirements, along with the corporation-wide policies and the general software visualization issues, were incorporated into the tool selection process. The next section provides an overview of an example tool selection.

4. Tool Selection Example

The evaluation of the visualization capabilities of the candidate tools was based on an existing framework for assessing the capabilities of reverse engineering tools [5], which was suitably augmented to address the constraints imposed by the project's singular requirements. The framework has previously been used in the application domains of traditional legacy system reengineering [8], reverse engineering Web site content and structure [7], and understanding net-centric applications [6]. This project represented the first time the framework was used in the context of embedded, real-time control systems.

The final choice by the industrial partner of Microsoft Visio [3] over several other commercial tools is illustrative of the issues facing software visualization tool developers and researchers hoping to have their work adopted by industry. Microsoft's Visio program is part of their successful Office family. However, it must be purchased as a separate product; it is not yet considered as crucial component of Office in the same way as say Word or PowerPoint are.

Visio is a business and technical drawing and diagramming program. It is designed to convey information visually. It has templates that setup the page appropriately and open stencils that contain predawn shapes. Visio is representative of "editable" graph visualization tools for both a graphics drawings and a graphics browsing.

Visio is mostly used to create images such as block diagrams, flowcharts, tables, floor plans, project schedules, and so on. It uses text and symbols to convey information at a glance. Its purpose is not to be a software visualization tool per se, given its limited layout choices. However, Visio was considered to be a good candidate software visualization tool for the project for several reasons:

- The industrial partner uses Visio in their current develop process. Therefore, they are familiar with its interface and capabilities. Deployment costs would be lowered substantially.
- The industrial partner already has its own set of symbols and icons used to represent their software and hardware artifacts. These icons come from the real-time design tool they use. The ability to use the same icons to represent recovered artifacts from the industrial partner's source code provides the potential for round-trip reverse engineering.
- Our reverse engineering toolset included data gathering engines to extract artifacts from source code and assorted types of documentation. The resultant information is stored in a central database. The database can be queried and updated by visualization tools such as Visio using standard application program interfaces. In this manner, Visio can be used to create diagrams in an automated manner, diagrams that are up to date with respect to the source code of the subject system.

The graph generated by Visio can be saved as a file (static), navigated by the user (with the appropriate Visual Basic code written to provide user interaction), and edited by the user. Therefore, Visio can be considered the most powerful of all the tools considered in our selection process. Indeed, it seemed to be the favorite choice of the industrial partner as well.

5. Conclusions

This paper discussed issues related to selecting software visualization tools for program understanding in an industrial context. Selecting the proper software visualization tool to produce such graphical documentation for embedded, real-time control systems in an industrial context is quite challenging.

The general issues related to software visualization must be addressed during tool selection in any application domain. For example, considering which types of graphs the tool supports. Static graphs are useful for inclusion in printed documentation that may accompany source code, or as part of an online documentation package. Interactive graphs lend themselves to hypertext navigation, and are quite common in Web-based code browsing systems. Editable graphs are the most complex of the three categories, but they do permit a closer connection between the reverse engineering process that produces the graphs and the regular forward engineering activities that are part of the overall development process.

Beyond the general software visualization issues, we also had to consider corporation-wide policies regarding tool usage, adoption issues, and process integration. These stringent policies precluded the selection of non-commercial tools, some of which may in fact have been the better selection in other circumstances. Based on our experience with this project, there is little doubt that adoption is *the* area where software visualization tool developers and researchers need to spend more effort. The barriers to tool adoption in an industrial setting are many and varied. There really is a quantum difference between a tool that works in the lab and a tool that works across an enterprise. Fortunately, the community seems to have begun to address this issue [4].

The project-specific requirements were the final sieve through which our candidate tools were filtered. The unique characteristics of embedded, real-time control systems mandated certain types of information that had to be extracted from the system, analyzed, and subsequently visualized. For example, the visualization of timing dependencies between low-level variables and high-level tasks is something that many tools do not support, at least not in the manner that the engineers are used to from their design tools.

The area of real-time, embedded control systems is a rich source of new challenges to the program understanding community. Hopefully this work will in some small way contribute to the field by identifying some of the key issues related to software visualization tools in this context. There are many opportunities for both academic researchers and commercial tool developers to explore in this area.

References

[1] Citrix Inc. "Citrix MetaFrame XP for Windows." Online at www.citrix.com/products/metaframexp.

[2] Hartmann, J.; Huang, S.; and Tilley, S. "Documenting Software Systems with Views II: An Integrated Approach Based on XML." *Proceedings of the 19th Annual International Conference on Systems Documentation* (SIGDOC 2001: Santa Fe, NM; October 21-24, 2001), pp. 237-246. ACM Press: New York, NY, 2001.

[3] Microsoft Corp. "Visio: The Office Business Diagramming Solution." Online at www.microsoft.com/office/visio.

[4] Müller, H. CASCON 2001 Workshop on Adoption-Centric Tool Development. November 7, 2001; Toronto, Canada.

[5] Tilley, S. *A Reverse-Engineering Environment Framework* (CMU/SEI-98-TR-005). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.

[6] Tilley, S. and DeSouza, M. "Spreading Knowledge about Gnutella: A Case Study in Understanding Net-Centric Applications." *Proceedings of the 9th International Workshop on Program Comprehension* (IWPC 2001: May 12-13, 2001; Toronto, Canada), pp. 189-198. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[7] Tilley, S. and Huang, S. "Evaluating the Reverse Engineering Capabilities of Web Tools for Understanding Site Content and Structure: A Case Study." *Proceedings of the 23rd International Conference on Software Engineering* (ICSE 2001: May 12-19, 2001; Toronto, Canada), pp. 514-523. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[8] Tilley, S. *Discovering DISCOVER* (CMU/SEI-97-TR-012). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, October 1997.