ICSE Workshop on Software Visualization

Wim De Pauw
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne NY 10532
USA
wim@us.ibm.com

Steven P. Reiss
Dept. of Computer Science
Brown University
Box 1910
Providence RI 02912
USA
spr@cs.brown.edu

John T. Stasko
College of Computing
GVU Center
Georgia Inst. of Technology
Atlanta GA 30332-0280
USA
stasko@cc.gatech.edu

Abstract

This workshop looks at current work in the area of software visualization with an emphasis on software understanding through visualization. It explores new visualization techniques, addressing software problems through visualization, frameworks for gathering and analyzing data for software visualization, software visualization systems, and experiments and experiences with software visualization. In addition to providing an overview of current research in the area, it provides a forum for discussions and cooperation among researchers in this and related areas.

1. Introduction

Software visualization is the use of pictures for looking at and understanding software systems. Today's software systems are inordinately complex and hence difficult to understand. Yet such systems still need to be designed, written, and maintained. Understanding the design, code, and behavior of software systems is thus an essential and important problem. Humans are inherently visual creatures, relying most on their sense of sight. It is only natural then that we attempt to draw pictures as a means of understanding our software systems. This is the motivation for software visualization.

2. Visualizing static aspects of a program

Contemporary programming languages embody countless lessons learned over nearly a half century of modern computing. In general, programmers use a language to map their ideas into a program space. From this perspective, visual tools can help to better understand and manipulate the mapping into the program space. Such tools complete a feedback loop that lets a programmer view and modify the mapping to suit his or her needs. Typically these visualizations are various forms of design diagrams. Today, much of this has been standardized into the various diagram classes supported by UML. However UML only addresses a small fraction of the overall structure of a software system and says very little about its behavior.

3. Visualizing dynamic aspects of a program

This concept of a mapping between the programmer's ideas and a program space is also useful because it has a direct analog in characterizing the dynamic aspects of a program. A program's dynamic behavior is just as important to its design, implementation, and refinement as its static specification.

This is especially true of object-oriented programs, where the gulf between static specification and run-time behavior is particularly wide. This has numerous causes: the dichotomy between the code structure as hierarchies of classes and the execution structure as networks of objects; the atomization of functionality - small chunks of functionality dispersed across multiple classes; and the sheer number of classes and complexity of relationships in applications and frameworks. But while much is known about the static aspects of programs, much less is known about characterizing and manipulating their dynamic aspects. Visualizing the execution of these programs is

0-7695-1050-7/01 $10.00 © 2001 IEEE

critical for understanding, debugging and improving their performance.

4. A wide spectrum of research directions

Considerable research [1] has been done over the past twenty years aimed at using visualization to provide programmers with a wide range of information about their systems. Today, with more powerful computers, more available computer graphics, and more complex software systems, more researchers and users are interested in and working on software visualization. Ongoing research in software visualization is working on extending it in a number of directions.

Some researchers are working on developing new visualization techniques that provide a deeper insight into different aspects of software structure or behavior. Others are looking at various software understanding problems and attempting to address them using existing visualization techniques in new ways. Others are developing frameworks to gather and organize the data needed for understanding through visualization. Others are developing visualization systems that provide a variety of different visualizations over a common database. Others are developing visualization frameworks that make developing and using software visualizations easier. Others are undertaking a variety of experiments that measure the value of software visualization for different tasks.

Since complexity in software systems is growing significantly, purely textual means often are too overwhelming for the programmer. Software visualization however allows us to more easily understand and reverse engineer these systems. Another challenge of analyzing these more complex systems is to find performance bottlenecks. The focus of optimization used to be microscopic, such as making a *while* loop run 10% faster. Nowadays, optimization typically involves understanding long paths across components that are also repetitive. The gains of such optimizations typically can make transactions multiple times faster.

Modern languages take away some of the burden of memory management by offering automatic garbage collection. However, this feature can be a double-edged sword. Programmers may get the false impression that they do not have to worry about memory at all. Practice shows that *memory leaks* in these programs can be very difficult to fix. Visualizing the data structures of a program in an intelligent way is crucial to solve these problems.

Algorithm animation is yet another example of software visualization that has proven to be very useful in teaching computer algorithms. By giving concrete depictions to the abstractions and operations of algorithms, algorithm animation makes the algorithms more concrete and meaningful.

5. Need for a common forum

While significant work is being done by a variety of researchers, there is no common forum for presenting this work and for researchers to obtain an understanding of what others are doing in the field. Work on software visualization tends to be a small part of a variety of different conferences, such as IEEE Visual Languages, ACM PASTE workshop, ICSE, IEEE Visualization, and IEEE Information Visualization. However, it is not the focus of any of these. The result is that the work on visualization is scattered, many researchers do not know what others are doing, and there is little sharing of information and software. This workshop on software visualization provides the necessary common forum for researchers.

References

[1] Stasko, John, Domingue, John, Brown, Marc H. and Price, Blaine A. (editors), *Software Visualization: Programming as a Multimedia Experience*, MIT Press, Cambridge, MA, 1998.