CFB: A Call For Benchmarks - for Software Visualization

Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent Ohio 44242 USA
jmaletic@cs.kent.edu

Andrian Marcus
Department of Computer Science
Wayne State University
Detroit, MI 48202 USA
amarcus@cs.wayne.edu

Abstract

The paper argues for the need of a benchmark, or suite of benchmarks, to exercise and evaluate software visualization methods, tools, and research. The intent of the benchmark(s) must be to further and motivate research in the field of using visualization methods to support understanding and analysis of real world and/or large scale software systems undergoing development or evolution. The paper points to other software engineering sub-fields that have recently benefited from benchmarks and explains how these examples can assist in the development of a benchmark for software visualization.

1 Introduction

Recently, the development of benchmarks has been highlighted [15] as a means to increase the scientific maturity of a discipline. Sim et al [15] detail a number of fields in Computer Science and Software Engineering that have proposed benchmarks to further research and understanding of the fields.

With regards to reverse engineering and program analysis a recent benchmark on dealing with fact extraction [16] motivated a number of improvements on tools such as cppx [3]. Also, developing a benchmark for clone detection was recently discussed at the International Workshop on Program Comprehension 2003 with a main goal of formalizing the meaning of source code clones and the like.

A number of individuals have argued for the Software Visualization community to develop a standard benchmark to support the research in the field. This important issue was discussed at the ICSE'01 Workshop on Software Visualization, VISSOFT'02, and most recently at the ACM Symposium on Software Visualization (SoftVis'03).

In particular, our recent discussions with Stephan Diehl, general chair of SoftVis'03, and Margaret-Ann Storey, an organizer for VISSOFT'02 and '03, motivated us to develop a Call-For-Benchmarks in Software Visualization. We will motivate why this may be the best means of developing a benchmark (suite) for software visualization research. We feel there is a need for a suite of problems that address different aspects of software visualization and argue for this type of approach. Additionally, we will propose a set of guidelines to help organize this call.

2 Aspects of Software Visualization

The focus of the benchmark will be to exercise software visualization systems/tools/methods in light of their applications toward supporting industrial software development, maintenance, and evolution. In order to frame this task we define five dimensions of software visualization [6]. These dimensions reflect the why, who, what, where, and how of the software visualization. The dimensions are as follows:

- Tasks – *why* is the visualization needed?
- Audience – *who* will use the visualization?
- Target – *what* is the data source to represent?
- Representation – *how* to represent it?
- Medium – *where* to represent the visualization?

These dimensions define a framework capable of accommodating a large spectrum of software visualization systems. This viewpoint subsumes such diverse topics as program visualization, algorithm animation, visual programming, programming by demonstration, software data visualization, and source code browsers. This diversity is reflected in the taxonomic descriptions of the field by researchers such as Price [9, 10], Roman [14], Myers [8], and Stasko [17].

Foremost, the benchmark should highlight different types of tasks. For instance one could propose a benchmark with the task of visualizing possible ADTs in legacy code or visualizing the run time activation of classes over a system. These are specific tasks that require (possibly) very different visualization metaphors and tools.

Before we continue this discussion let us present a general reference model for information visualization.

This will help focus the particulars of the benchmark with regard to the underlying pre-processing and analysis that must accompany any software visualization tool or method.

3 A Reference Model for Visualization

Card [1] proposes that visualization is a mapping from data to a visual form that the human perceives. Figure 1, adapted from [1], describes these mappings and serves as a simple reference model for visualization. In this figure, the flow of data goes through a series of transformations. The human may adjust these transformations, via user controls, to address the particular application task.

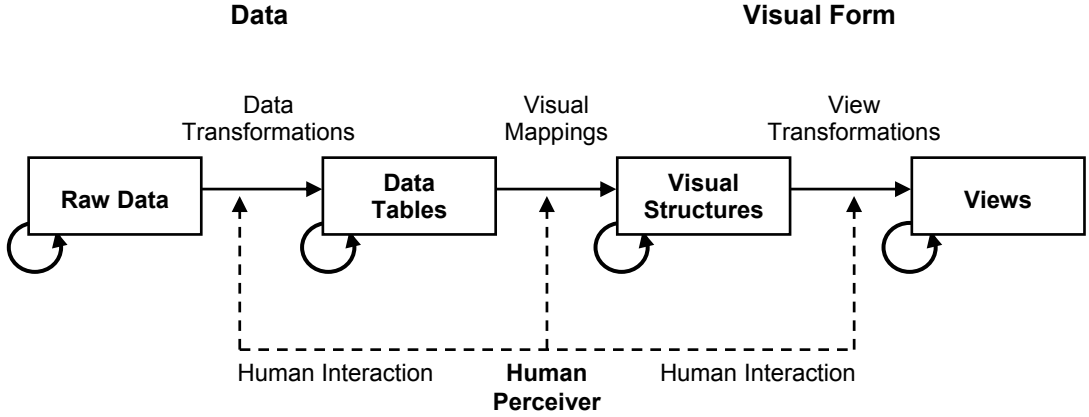
The first transformation converts raw data into more usable data tables. The raw data is typically in some domain specific format that is often hard, or impossible, to work with. This is very apparent when working with trace data generated from program executions. Data tables [1] are relational depictions of this data. Information about the relational characteristics of the data (meta data) can also be included in the data tables. Meta data is descriptive information about the data [19]. From here, visual mappings transform the data tables into visual structures (graphical elements). Finally, the view transformations create views of the visual structures by specifying parameters such as position, rotation, scaling, etc. User interaction controls the parameters of these transformations. The visualizations and their controls are all with respect to the application task.

The core of the reference model is the mapping of a data table to a visual structure. Data tables are based on mathematical relationships whereas visual structures are based on graphical properties processed by human vision. Although raw data can be viewed directly, data tables are a vital intermediate step when the data is abstract [2, 5, 12].

Software visualization maps to this reference model directly. The raw data is source code, execution data, design documents, etc. In the case of execution (trace) data, the readability is minimal. However, source code is readable, at least on a small scale, that is, one can hardly keep in mind more than a few dozen lines of source at one time. Data tables, an abstraction of the raw data, take the form of abstract syntax trees, program dependence graphs, or class/object relationships for example. A variety of software analysis tools can generate this type of data (table). Visual structures are then the software-specific visualizations we render. These visual structures are typically very specific to a particular software engineering task.

This model also points out the need to transform raw data into something more usable. This includes initial acquisition, quality, and granularity of the data. While these issues are not high profile for source code, they are a key component for dealing with the huge amounts of data that can be generated from execution traces, or from parse trees of large systems.

The software visualization process maps on top of this reference visualization model. Roman [14] and Price [9, 10], each define their own general model of the software visualization. Their views are more domain-specific and



Raw Data: idiosyncratic formats
Data Tables: relations (cases by variables) + meta data
Visual Structures: spatial substrates + marks + graphical properties
Views: graphical parameters (position, scaling, clipping, etc.)

Figure 1. Reference Model for Visualization. Visualization can be described as a mapping of data to visual form that supports human interaction for making visual sense [1].

omit aspects related to generation of views and data transformations. These models drive the definition of their respective taxonomies. We believe the general information visualization reference model should also be taken into direct consideration by a software visualization system designer.

4 Composition of the Benchmark

Given this general reference model we can now define benchmarks in terms of each of its specific components in conjunction with the task, audience, target, etc being addressed. A question that must be raised at this point is whether the underlying program/data/run time analysis methods are an integral part of the software visualization method? That is, can we (completely) decouple the visual structures and views from the underlying raw data and data tables? Obviously in general the answer to this question is no. However, the authors own work [7] along with others [18] counters this to some degree within a broad, abet limited, set of problem domains.

Of course, the concept that a software visualization tools is quite task specific and tightly coupled with the underlying data analysis is what makes construction of a single general benchmark, for software visualization, quite difficult (impossible). However, for a visualization tool to be widely utilized it should be interoperable with a variety of tools and environments.

This being the case, what then must the benchmark be composed of? We believe the general consensus is that a number of distinct problems (i.e., tasks, target, and audience) of differing domains, each with its own data set must be developed. The data set could include raw data but alternatively include data tables (or both). Providing data tables will drastically improve the ability to compare the visual aspects of methods as opposed to the underlying analysis methods.

Furthermore, the stated task of a given benchmark must be well directed at software engineering problems. We could easily fall into comparing 2D graph layout algorithms, whereas the real software visualization problem is more like the comparison of UML class diagrams layout methods (in a 2D space). Of course there must be an agreed upon quality measure. For class diagrams, recent work on the esthetics of UML diagram layout [4, 11] can help provide guidelines. In this case, the data table (UML class model) is all that is necessary.

Broader problems may include the visualization of cross cutting concerns within a given system. Here one must supply a system (raw data), with known aspects, and (hopefully) pointers to (or a list of) these aspects within the source (data tables). In this case the weaker the data table, the more of an analysis problem this becomes.

Development of a benchmark for visualizing the run time behavior of a system may be more difficult for some particular tasks. Providing an execution trace for a given system along with specific features of that trace that are deemed interesting is quite straight forward. However, developing a benchmark for visualizing the execution of a system in real time such as the research being done by Reiss [13] may be more difficult. However this could be posed as a specific question such as with debugging or bottleneck location. Of course the underlying analysis and data gathering is a permanent issue.

5 Call For Benchmarks

To develop a benchmark suite for software visualization we propose a Call For Benchmarks much like a Call For Papers. We issue this call to all researchers active and/or interested in software visualization. The plan is to collect all proposed benchmarks, review each, and have a round of revisions/clarifications. The collection will assembled and made available to the research community on the web. This should coincide with a related conference or workshop and the benchmark could be presented in a working session or the like to motivate individual research groups to apply the benchmarks to their work.

The goal is to collect the results of using the benchmarks and present the findings in a paper, presentation, and/or web site.

We, the authors, invite benchmark proposals. The submitted benchmarks should include:

- Description of the proposed benchmark
- Software engineering task being addressed
- The data (sets) necessary (source code, models, data tables, etc.)
- What types of data analysis are necessary (if any) to apply the benchmark
- An evaluation method
- Types of user interaction required

E-mail your benchmark proposal to both jmaletic@cs.kent.edu and amarcus@cs.wayne.edu. For further information visit the web site www.sdml.info.

6 References

[1] Card, S. K., Mackinlay, J., and Shneiderman, B., Readings in Information Visualization Using Vision to Think, San Francisco, CA, Morgan Kaufmann, 1999.

[2] Chi, E. H., Barry, P., Riedl, J. T., and Konstan, J., "A spreadsheet approach to information visualization", in Proceedings of Information Visualization Symposium '97, 1997, pp. 17-24,116.

[3] CPPX, "CPPX - Open Source C++ Fact Extractor", Web page, http://swag.uwaterloo.ca/~cppx/, 2001.

[4] Eichelberger, H., "Nice Class Diagrams Admit Good Design", in Proceedings of ACM Symposium on Software Visualization (SoftVis'03), San Diego, CA, June 11-13 2003, pp. 159-168.

[5] Levoy, M., "Spreadsheet for images", Computer Graphics, vol. 28, 1994, pp. 139-146.

[6] Maletic, J. I., Marcus, A., and Collard, M. L., "A Task Oriented View of Software Visualization", in Proceedings of 1st IEEE Workshop of Visualizing Software for Understanding and Analysis (VISSOFT'02), Paris, France, June 26 2002, pp. 32-40.

[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. to appear.

[8] Myers, B. A., "Taxonomies of Visual Programming and Program Visualization", Journal of Visual Languages and Computing, vol. 1, no. 1, March 1990, pp. 97-123.

[9] Price, B. A., Baecker, R. M., and Small, I. S., "A Principled Taxonomy of Software Visualization", Journal of Visual Languages and Computing, vol. 4, no. 2, 1993, pp. 211-266.

[10] Price, B. A., Baecker, R. M., and Small, I. S., "An Introduction to Software Visualization", in Software Visualization, Stasko, J., Dominque, J., Brown, M., and Price, B., Eds., London, England MIT Press, 1998, pp. 4-26.

[11] Purchase, H. C., "Effective information visualisation: a study of graph drawing aesthetics and algorithms", Interacting with Computers, vol. 13, no. 2, December 2000 2000, pp. 147-162.

[12] Rao, R. and Card, S. K., "Exploring large tables with the table lens", in Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'95), 1995, pp. 403-404.

[13] Reiss, S. P., "Visualizing Java in Action", in Proceedings of ACM Symboposium on Software Visualization (SoftVis'03), San Diego, CA, June 11-13 2003, pp. 57-66.

[14] Roman, G.-C. and Cox, K. C., "A Taxonomy of Program Visualization Systems", IEEE Computer, vol. 26, no. 12, December 1993, pp. 11-24.

[15] Sim, S. E., Easterbrook, S., and Holt, R. C., "Using Benchmarking to Advance Research: A Challenge to Software Engineering", in Proceedings of 25th International Conference on Software Engineering (ICSE'03), Portland OR, May 3-10 2003, pp. 74-83.

[16] Sim, S. E., Holt, R. C., and Easterbrook, S., "On Using a Benchmark to Evaluate C++ Extractors", in Proceedings of 10th International Workshop on Program Comprehension, Paris, France, 2002, pp. 114-123.

[17] Stasko, J. T. and Patterson, C., "Understanding and Characterizing Software Visualization Systems", in Proceedings of IEEE Workshop on Visual Languages, Seattle, WA, September 1992, pp. 3-10.

[18] 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.

[19] Tweedie, L., "Characterizing interactive externalizations", in Proceedings of Conference on Human Factors in Computing Systems (CHI '97), 1997, pp. 375-382.