A Qualitative and Quantitative Evaluation of Software Visualization Tools

Sarita Bassil and Rudolf K. Keller
Département IRO
Université de Montréal
C.P. 6128, succursale Centre-ville
Montréal, Québec H3C 3J7, Canada
+1 (514) 343-6782
{bassil, keller}@iro.umontreal.ca
http://www.iro.umontreal.ca/~{bassil, keller}

Abstract

Recently, many software visualization (SV) techniques and tools have become available to developers in order to help them understand their software. In our project, firstly, we identified a list of SV tools that we described, and evaluated qualitatively using the taxonomy of Price et al. Our experience with this taxonomy as well as the impact of the qualitative evaluation on Bell Canada will be discussed in this paper.

Then, a quantitative evaluation was conducted in spring 2000 with more than 100 participants. In this paper, we especially put the emphases on the perspectives of the results, by reporting on our experiences and by discussing the major findings and their impact.

Keywords: *software engineering, software visualization tool, qualitative evaluation, survey, software comprehension.*

1. Introduction

Today's software systems are increasingly large and complex. This makes the tasks of programming, understanding, and modifying the software more and more difficult. Therefore, tools for supporting these tasks have become essential. One key aspect of such support is software visualization (SV). Recently, a lot of SV techniques and tools have become available to support activities related to the software life cycle.

In the SPOOL (Spreading desirable Properties into the design of Object-Oriented, Large-scale software systems) project, an industry/university collaboration between Bell Canada and University of Montreal, we are investigating concepts and tools for assessing and improving the design of large software systems [6, 10]. To put this research into a broader perspective, and to provide Bell Canada with much-needed SV tools information, we set out to conduct a study on this kind of tool.

We decided to firstly identify a number of SV tools that we evaluated qualitatively using the taxonomy of Price *et al.* [9]. Then, we questioned the users of SV tools about their perceptions on what has worked and what has not worked when applying a specific tool.

We feel that this research and the obtained results will be useful beyond the SPOOL project in at least four ways. First, the qualitative evaluation carried out can help the interested people by giving them an example on how to use a particular taxonomy to evaluate a specific SV tool. Then, the wish lists and problems expressed by the participants of the survey will give SV tool builders valuable input for their products. Furthermore, prospective buyers and users of SV tools can consult

This research was supported by the SPOOL project organized by CSER (Consortium for Software Engineering Research) which is funded by Bell Canada, NSERC (National Sciences and Research Council of Canada), and NRC (National Research Council of Canada).

the two lists of the more than 50 tools covered by the study and use the questionnaire to conduct their own evaluations.[1] Finally, the shortcomings and limitations of current SV technology as revealed by the survey may define an agenda for further research in the domain.

In the remainder of this paper, we first discuss the qualitative evaluation, by summarizing the most interesting points. Then, in Section 3, we summarize the quantitative evaluation and discuss the major findings and their impact. The final section provides a conclusion and an outlook into future work.

2. Qualitative evaluation of SV tools

2.1. Description of SV tools

Considering a number of SV tools to evaluate help us to identify, understand and compare the properties and characteristics of these tools. For this reason, we identified a list of 30 SV tools among which we retained seven that we described then evaluated using the taxonomy of Price *et al.* Four main criteria were applied to retain those seven tools. Indeed, firstly, we wanted to evaluate academic as well as commercial tools. Then, the tools selected should be interesting for reverse engineering activities; whereas the third and the fourth criterion respectively related to the availability of the tools, and the interest granted to these tools within the software quality assessment team of Bell Canada. Taking into account these criteria, we tried to identify a number of tools representative of the 30 found. We came up with three commercial tools (Sections 2.1.1, 2.1.2, 2.1.3) and four academic (Sections 2.1.4, 2.1.5, 2.1.6, 2.1.7). In the Sections below, we will give a quick description of these tools (for details refer to [1]).

2.1.1. Software understanding system (xSuds) [14]. It is made of seven tools (xAtac, xRegress, xVue, xSlice, xProf, xFind, xDiff), which make it possible to understand, debug, test, maintain, and analyze software written in C or C++. The user of xSuds can visualize with color, through these various tools, the software source code.

2.1.2. Tom Sawyer products [17]. They are composed of two principal tools: the Graph Layout Toolkit (GLT) and the Graph Editor Toolkit (GET). The GLT is considered as being the visualization tool, while the GET is a graph editor, which provides advanced edition functions.

2.1.3. Discover [5]. It is an information system, which supports the two programming languages C and C++. It is composed of five tools (Develop/set, CM/set, Reengineer/set, Doc/set, Admin/set) and a number of applications for the comprehension and the development of large-scale software. The tool Develop addresses more than the others the software comprehension.

2.1.4. Visualizing Graphs with Java (VGJ) [7]. It is a tool of graphic layout and edition of graphs. It supports an input and an output of files in GML (Graph Modeling Language), which is considered as an increasingly widespread graph specification language.

2.1.5. daVinci [15]. It is a SV tool for directed graphs, similar to VCG (Section 2.1.7). daVinci takes as input a file in TR format (Term Representation).

2.1.6. Swan [16]. It is a data structures visualization system. The visualized software should be written in C or C++, and will be annotated by calls to a library, which controls the sequence of visualization. This annotation will then be displayed via a viewer.

2.1.7. Visualization of Compiler Graphs (VCG) [18]. It reads a textual specification of a graph (GDL format - Graph Language Description) and allows visualizing it using layout algorithms. VCG is intended to visualize graphs, which are automatically produced by programs. An example of such programs would be Call Graph Drawing Interface (CGDI) [3].

[1] The questionnaire and a complete list of the names and web references of the SV tools for which we received answers to the questionnaire, as well as the list related to the qualitative evaluation can be found from <http://www.iro.umontreal.ca/labs/gelo/sv-survey/>.

2.2. Evaluation of SV tools

After identifying and describing the seven SV tools, we evaluated them to illustrate their characteristics. We used an existing taxonomy, that of Price *et al.*, for the evaluation. This taxonomy is considered as the most complete among the studied taxonomies [8, 9, 11, 12]. It suggests a hierarchy of six categories. Each one of these categories is broken up into subcategories. These subcategories are presented as questions and form the different branches of the hierarchy.

The *Scope* category describes the interval of software that can be visualized by the SV tool under consideration. The *Content* category describes the subset of information about the visualized software. The most significant element from the point of view of the user is the output of the visualization, which is described in the *Form* category. This category addresses the parameters and the limitations that govern the output. The *Method* category characterizes the elements related to the specification of the visualization. The *Interaction* category addresses the techniques used by the observer (user of the SV tool) to control and interact with the SV tool. Finally, the effectiveness of the SV tool is determined by the *Effectiveness* category. Empirical evaluations are necessary for this category.

Considering our evaluation of the seven SV tools, several elements were observed with respect to each one of these categories. The following Sections present the essential points for each category (for details refer to [1]).

2.2.1. *Scope* category. We notice that all the studied tools can handle a generalized interval of visualized software. As regarding the programming language of the software, we realize that the tools taking as an input a particular file format, accept software written in any programming language. The corresponding tools are VGJ and daVinci. In the case of VCG, the use of CGDI limits the input to C or C++ files. No restriction applies to the application domain of the visualized software. However, there is a certain kind of software that is particularly good to visualize (e.g., directed graphs for daVinci, graphs in general for Swan, and programs with a lot of function calls for VCG (taking into account the CGDI)). In respect to scalability, the evaluated commercial tools do not expose theoretically any problem of scalability. On the other hand, for the majority of the academic tools, the problem of scalability exists.

2.2.2. *Content* category. The first branch (program) of this category corresponds to the visualization of the source code. We notice that the more a SV tool makes it possible to visualize software with a low level of abstraction (e.g., source code), the higher is the level of fidelity. The tools that have in an obvious way this property are xSuds and Discover. Swan is a particular case for the *program* property. The evaluation of this tool with respect to this property, depends on the annotation. The data gathering to create visualization is done for the majority of the tools (xSuds, VGJ, daVinci, Swan and VCG) during compilation. Three of these tools (xSuds, Swan and VCG) collect also information during execution time.

2.2.3. *Form* category. All the evaluated tools offer graphic views, except xSuds that is centered on the source code. All the tools have the color property. Even for daVinci, which initially did not offer the possibility to visualize in color, this option was added by the Visualization Research Group [19]. The majority of the tools does not support the 3D, neither the animation, nor the sound. The coarse-granularity is often possible by the *overview windows* that are offered by the majority of the tools. The possibility of temporarily hiding the information that does not have an immediate interest is an important characteristic, but unfortunately it is not always supported. The two properties *multiple views* and *synchronization* are only supported by xSuds, and Discover.

2.2.4. *Method* category. The specification of the visualization can be automatically done by writing program taking as an input a specific format and giving as an output a file of format required by the tool. For instance, xSuds takes as an input an *atac* file format. daVinci accepts a file of particular format (TR). Same thing for VGJ, for which GML format is necessary. Swan requests a preliminary knowledge of the code and provides the SAIL (Swan Annotation Interfaces Library) used to annotate the source code. VCG requires a GDL format for which we found CGDI, which makes it possible to pass from a C or C++ file to a GDL file. The seven evaluated tools do not offer any intelligence; on the other hand all the tools make it possible to particularize the visualization.

2.2.5. *Interaction* category. The majority of the tools provide the same style (buttons, menus, etc.) to receive instructions from the user. Swan offers also a command line. As regarding the navigation, all the tools, except VCG, allow navigation across the visualized software.

2.2.6. *Effectiveness* category. The clearness of the visualization is a subjective property. We should evaluate if the visual metaphors used are intuitive. On the other hand, the two properties: *empirical evaluation* and *use in the production* are more

easily evaluated. No empirical evaluation was carried out for the seven studied tools. Three of these tools are commercial (xSuds, Tom Sawyers products and Discover) and, consequently, they are used in production. The four remaining tools are academic and are not used in a widespread way in production.

2.3. From qualitative to quantitative evaluation

The study that we have just presented enabled us to evaluate and understand the tools within a formal framework. It provided Bell Canada with useful qualitative information about commercial as well as academic SV techniques and tools. In fact, the software quality assessment team of Bell Canada became aware, via our research, of a great number of SV tools characteristics that an almost complete taxonomy may be looking for in such tools. In addition, the team showed a remarkable interest to the list of 30 SV tools identified, and particularly to the seven tools evaluated; it became well informed about the availability, the usefulness of the technical aspects and the relevance of these tools, which may be a possible alternative to the current SV tools used by the team and becoming less and less available.

In spite of all the positive elements stated before, we would like to specify some weak points of our qualitative study. In fact, most time the evaluation of the properties related to the taxonomy of Price *et al.* was done in a subjective way. Moreover, no cognitive aspects were considered during this evaluation. In addition, this study was limited to evaluate a set of only seven tools. Although they are representative, these tools remain a very tiny sample taking into account the great number of existing SV tools. As a remedy, a quantitative evaluation of SV tools was conducted. This survey made it possible to collect points of view of a great number of users working on different SV tools. The following section will thus present in a brief way the major findings of this survey and their impact. More details about the survey results can be found in [1, 2].

3. Quantitative evaluation of SV tools

A survey on SV tools was conducted between March and May 2000. It addresses various functional, practical, cognitive as well as code analysis aspects that users may be looking for in SV tools.

In general, the participants were quite pleased with the SV tool they were using. Functional aspects such as searching and browsing, and use of colors were rated as the most essential aspects. Hierarchical representations as well as navigation across hierarchies were also strongly desired, especially for object-oriented software. On the other hand, source code visualization was almost always wished when the visualized software was procedural. The three aspects that were least appreciated, especially in industry, were animation, 3D visualization and VR techniques. However, animation was identified as being quite useful for declarative software.

Regarding the practical aspects of SV tools, we found that tool reliability was classified as the most important aspect, followed by the ease of using the tool and the ease of visualizing large-scale software. Unfortunately, we found a disturbing gap between the high importance attached to the two aspects ease of use and quality of the user interface, and the ratings of these qualities in the SV tools in practical use. Tool documentation was also estimated as very important. Furthermore, the participants specified an interesting list of additional desired aspects. Many of these aspects related to the integration of SV tools into other tools, such as code generators, design partitioning tools, editors, metrics tools, generators and managers of documentation, schedulers, etc.

Clearly, we verified that code comprehension is considered key for carrying out various maintenance and software life cycle tasks. Moreover, a number of interesting patterns concerning the benefits and the improvements of SV tools could be derived.

Concerning code analysis aspects, it seems that only a low number of these aspects are supported by current SV tools. Among 24 different aspects, only three were identified as being supported by more than half of the tools for which we received answers. These three aspects were: visualization of function calls, of inheritance graphs, and of different levels of detail in separate windows. Aspects related to the calculation of metrics were the least supported, but were sometimes desired.

From this survey, we can tell that today's SV tools have many uses, notably in the software comprehension process. The users specified many benefits from using a particular SV tool. The two most quoted ones were the savings in time when carrying out a specific task, and better comprehension of the visualized software. However, numerous desired aspects and improvements to the SV tool in use were also indicated by the participants. These are clearly issues that future SV tools, and even future versions of today's tools, would be expected to handle. Consequently, the challenge for future SV tool builders is to provide tools which integrate third-party tools, make it possible to import from/export to other SV tool formats, and are

more customizable in terms of compiler settings. A more complete list of the improvements that should be done to SV tools is presented in [1, 2]. These items not only concern tool builders, but also constitute issues of a research agenda.

4. Conclusion and future work

We began this work by identifying seven tools that we described. These tools were evaluated thereafter using the taxonomy of Price *et al.* considered as being the most complete among the number of taxonomies that we studied. However, it does not take into account cognitive aspects of SV tools, which were addressed in an exclusive way in the taxonomy of Storey *et al.* [13]. Consequently, we think that a taxonomy which gathers at the same time cognitive as well as concrete aspects of SV tools would be desirable. In brief, the qualitative evaluation allowed us to evaluate in a formal way a number of SV tools. However, some weak points related to this evaluation were stated in Section 2.3. To overcome these limitations, a survey (quantitative evaluation) on SV tools was conducted. It made it possible to collect the point of view of users working on various SV tools. Several interesting results were generated from this survey (Section 3).

There are two main directions in which we plan to continue this research. For one thing, via multi-phase survey techniques [4], we aim to identify representative target populations for different survey questions and we will attempt to test significant support of hypotheses. As a second direction, we want to carry over SV techniques and tools found and empirically evaluated during the qualitative and quantitative phases of the project, into Bell Canada's assessment process, in order to evaluate their positive impact and limitations. We will also study how visualization may guide quantitative assessment of software.

References

[1] Bassil, S., Qualitative and Quantitative Evaluation of Software Visualization Tools. *Master's thesis*, Université de Montréal, Montreal, Quebec, Canada, Dec. 2000. French title: Évaluation Qualitative et Quantitative d'Outils de Visualisation Logicielle.

[2] Bassil, S. and Keller, R. K., Software Visualization Tools: Survey and Analysis. In *Proceedings of the 9th International Workshop on Program Comprehension (IWPC'01)*, Toronto, Ontario, Canada, May 2001. IEEE. To appear.

[3] Call Graph Drawing Interface Page. On-line at <http://www.ida.liu.se/~vaden/cgdi/>.

[4] Cooper, D. R. and Schindler, P. S., Business Research Methods. *Irwin/McGraw-Hill*, sixth edition, 1995.

[5] Discover Information Set. On-line at <http://www.upspringsoftware.com/products/discover/index.html>.

[6] Keller, R. K., Schauer, R, Robitaille, S., and Pagé, P., Pattern-Based Reverse Engineering of Design Components. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, pp. 226-235, Los Angeles, CA, May 1999.

[7] Manual for VGJ. On-line at <http://www.eng.auburn.edu/department/cse/research/graph_drawing/manual/vgj_manual.html>.

[8] Myers, B. A., Taxonomies of Visual Programming and Program Visualisation. *Journal of Visual Languages and Computing*, volume 1, pp. 97-123, 1990.

[9] Price, B. A., Baecker, R. M., and Small, I. S., A Principled Taxonomy of Software Visualisation. *Journal of Visual Languages and Computing*, volume 4, pp. 211-266, 1993.

[10] Robitaille, S., Schauer, R., and Keller, R. K., Bridging Program Comprehension Tools by Design Navigation. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, pp. 22-32, San Jose, CA, Oct. 2000.

[11] Roman, G.-C. and Cox, K. C., A Taxonomy of Program Visualisation Systems. *IEEE Computer*, 26(12), pp. 11-24, Dec. 1993.

[12] Stasko, J. T. and Patterson, C., Understanding and Characterizing Software Visualisation Systems. In *Proceedings of the IEEE 1992 Workshop on Visual Languages*, pp. 3-10, Seattle, WA, 1992.

[13] Storey, M.-A. D., Fracchia, F. D., and Müller, H. A., Cognitive Design Elements to Support the Construction of a Mental Model During Software Exploration. *Journal of Systems and Software*, volume 44, pp. 171-185, Jan. 1999.

[14] Telcordia Software Visualization and Analysis Toolsuite (xSuds). On-line at <http://xsuds.argreenhouse.com/>.

[15] The Graph Visualization System daVinci. On-line at <http://www.informatik.uni-bremen.de/daVinci/>.

[16] The Swan User's Manual. On-line at <http://geosim.cs.vt.edu/Swan/manual.pdf>.

[17] Tom Sawyer Software: Products and Solutions documentation set. On-line at <http://www.tomsawyer.com/literature/index.html>.

[18] VCG Overview. On-line at <http://www.cs.uni-sb.de/RW/users/sander/html/gsvcg1.html>.

[19] Visualisation Research Group. On-line at <http://www.dur.ac.uk/~dcs0elb/visual/>.