Software Visualization Tools: Survey and Analysis

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. There is ample anecdotal evidence that appropriate visualization can significantly reduce the effort spent on system comprehension and maintenance, yet we are not aware of any quantitative investigation and survey of SV tools. This paper reports on a survey on SV tools which was conducted in spring 2000 with more than 100 participants. It addresses various functional, practical, cognitive as well as code analysis aspects that users may be looking for in SV tools. The participants of the survey rated the usefulness and importance of these aspects, and came up with aspects of their own. The participants were in general quite pleased with the SV tool they were using and mentioned various benefits. Nevertheless, a big gap between desired aspects and the features of current SV tools was identified. In addition, a list of improvements that should be done to current tools was assembled. Finally, the collected data tends to suggest that in general, code analysis aspects were not highly supported by the tools.

Keywords: *software visualization tool, survey, questionnaire, software comprehension, source code analysis.*

1. Introduction

Today's software systems are increasingly large and complex, and their development and maintenance typically involves the collaboration of many people. This makes the tasks of programming, understanding, and modifying the software more and more difficult, especially when working on other people's code. 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 such as analysis, modeling, testing, debugging, and maintenance. There is ample *anecdotal* evidence that appropriate visualization can significantly reduce the effort spent on these activities. Harel, for instance, states that "using appropriate visual formalisms can have a spectacular effect on engineers and programmers." [11]

A number of taxonomies for SV [16, 18, 23, 29, 30] have been suggested. Some of these taxonomies have been used to characterize and informally assess SV tools. Furthermore, a number of experiments have been performed on users of SV tools. Storey *et al.* [31], for instance, observed users when solving program understanding tasks, and they tried to describe how three tools (Rigi [21], SHriMP [25], and SNiFF+ [26]) supported or hindered a number of comprehension strategies. Yet, we are not aware of any quantitative investigation and survey of SV tools.

In the SPOOL (Spreading desirable Properties into the design of Object-Oriented, Large-scale software systems) project, an industry/university collaboration between the software quality assessment team of Bell Canada and the software engineering group at University of Montreal, we are investigating concepts and tools for assessing and improving the design of large software systems. This involves the development of a reverse engineering environment which comprises various analysis and visualization tools and supports software comprehension by its organization around patterns [12, 22, 24]. To put this research into a broader perspective, and to provide Bell Canada with

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).

much-needed survey information, we set out to conduct a study on SV tools.

We decided to question the users of SV tools about their perceptions on what has worked and what has not worked when applying a specific tool. Using existing taxonomies and addressing technical as well as cognitive aspects of SV tools, we created a catalog of properties of such tools. Then, we tried to arrange these properties into a number of questions forming a questionnaire of SV tools, with the principal interest, as specified by Bell Canada, of identifying desirable and undesirable properties of SV tools, and evaluating their capabilities for reducing the effort of software comprehension and design assessment. We feel that this research and the obtained results will be useful beyond the SPOOL project in at least three ways. First, the wish lists and problems expressed by the participants of the survey will give SV tool builders valuable input for future versions of their products. Furthermore, prospective buyers and users of SV tools can consult the list of the more than 40 tools covered in the survey[1] and use the questionnaire[2] to conduct their own evaluations. 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 present the methodology used to conduct the survey, by describing the questionnaire and the method of data collection and analysis. In Section 3, we detail the survey results related to the functional, practical, and cognitive aspects of SV tools. Section 4 presents some specific technical information about SV tools, with a focus on code analysis support. The final section puts the survey into perspective, by reporting on our experiences, by discussing the major findings and their impact, and by providing an outlook into future work.

2. Survey methodology

2.1. Design of the questionnaire

Similarly to Watson *et al.* [41] and Price *et al.* [18], we adopted, as a basis for the questionnaire, quite a general definition of SV tool. By SV tool we understand a tool that supports the comprehension of software systems by means of graphical and/or textual representations.

The questionnaire is organized into two parts, where Part I is intended for any SV tool user, and Part II calls on expert users of SV tools.

[1] A complete list of the names and web references of the SV tools for which we received answers to the questionnaire can be found at <http://www.iro.umontreal.ca/labs/gelo/sv-survey/list-of-tools.html>.

[2] The questionnaire is available on-line, in English or in French, at <http://www.iro.umontreal.ca/labs/gelo/sv-survey/questionnaire.html>.

The first part of the questionnaire comprises six sections with a total of 21 questions. In the first two sections (Sections 3.1 and 3.2 in this paper), information about the participants' context of work and the software systems being visualized is requested. Then, questions about the functional and practical aspects (desired or not) of SV tools are asked (Sections 3.3 and 3.4). The last two sections of Part I concern the "SV tool at hand", that is, the participants are invited to answer these sections in respect to a particular SV tool of their choice, typically the one with which they are most familiar. In this way, a cognitive evaluation of the SV tool at hand is carried out (Section 3.5), together with an assessment of the benefits resulting from the use of that tool, as well as the improvements that should be done to it (Section 3.6).

Part II of the questionnaire addresses expert users of SV tools and consists of two sections with an overall of 17 questions. It refers entirely to the "SV tool at hand". The first section (Section 4.1) aims to collect technical information about the SV tool. In the second section (Section 4.2), questions concerning the support of code analysis by the SV tool are asked.

Note that most of the questions in the two parts are closed, giving a certain number of pre-defined answers from which the participants can choose. However, in order to allow the participants to detail their own answers, empty fields are provided for some of the questions, and furthermore, two open-ended questions are presented at the end of Part I. These two questions constitute the last section of Part I, where the participants' opinion about the benefits and improvements of the SV tool at hand is sought.

A number of elements related to survey design [5] were taken into account when designing the questionnaire. Furthermore, we leveraged the experience reported in [13, 14, 15]. A draft of the questionnaire was circulated within our group and sent to some of our colleagues in CSER, the consortium in which SPOOL participates. Subsequently, a number of useful comments were incorporated in the final version of the questionnaire.

2.2. Data collection and analysis

An invitation was circulated on various electronic mailing lists (SIGCHI, Software Engineering Canada, Psychology of Programming Interest Group, Reverse Engineering, etc.) and newsgroups (comp.lang.visual, comp.software-eng, comp.lang.java, comp.lang.c++, ibm.software.vajava.ide, etc.). Moreover, a message was sent explicitly to some known users of specific SV tools, such as daVinci [34], Discover [6], GraphViz [9], Grasp [10], PBS [17], Rigi [21], SHriMP [25], TkSee [36], Tom Sawyer Software [38], VCG [40], xSuds [33], etc.

The responses were all received between March and May 2000. The sample consists of 107 participants for Part

I (100 participants have filled out the English version, and 7 have filled out the French one), of which 41 (37 for the English version and 4 for the French version) have filled out Part II as well.

To analyze the collected data, we have applied standard statistical techniques, including generation of frequencies (percentages) and means, calculation of correlations using Kendall's tau-b or tau-c, and verification of hypotheses using t-test values. Calculations were done using the SPSS package [28].

3. Survey results of Part I

3.1. Distribution of the participants

The questionnaire was especially designed for SV tool users from industry, and indeed two thirds of the participants were working in companies and the rest in a research setting. Considering their profession, we note that again two thirds fitted into the category "development and maintenance", whereas the remaining third were involved in management or research. The distribution of participants according to the size of their workgroup was as follows: 35% of the participants were working in a group of less than five persons; 36% were in a group of five to 20 persons, and 17% were involved in big projects with a group of more than 20 persons. The remaining 12% didn't answer the question related to workgroup size.

60% of the participants indicated that more than one SV tool were used in their workgroup. This suggests that no single tool in these workgroups provides all desired characteristics, and that complementary SV tools are being used. Furthermore, the majority of the tools in these workgroups were commercial. More than three quarters of the participants estimated to have a good knowledge of the SV tool at hand. A similar rate was found for the knowledge of the visualized software systems, that is, for the familiarity with their source code. Not surprisingly, there is a significant positive correlation between the knowledge level of the visualized software systems and that of their application domain (tau-b = 0.458) and their source code language (tau-b = 0.489).

3.2. Characteristics of the visualized software systems

A high number of the visualized software systems (VSS) were developed using an object-oriented language (73 systems), against 48 systems using a procedural language, and five systems using a declarative language such as Prolog. The total of these numbers (126) is higher than the number of participants (107), since the participants could indicate several languages. Taking into account the application type of the VSS and proposing to the participants nine different answers from which they can choose, we found that the most frequent type was real-time (21%), followed by scientific/engineering (16%), and information system (15%).

In respect to their size, the VSS considered by the participants were evenly spread according to the three categories small, medium, and large. In fact, one third of the VSS were large systems with more than 1000 KLOC, one third were systems of medium size (between 100 and 1000 KLOC), and the remaining third were small systems with less than 100 KLOC. Not surprisingly, we found a positive correlation between the size of the VSS and the size of the workgroup (tau-b = 0.250).

We noticed that a large number of participants (42%) specified that the VSS were complex, 36% estimated that they were of average complexity, and 18% found they were easy, with the remaining participants (4%) leaving that question unanswered. Quite similar percentages were found for the complexity level of the tasks carried out on the VSS. In fact, a significant positive correlation exists between these two variables (tau-b = 0.539); this suggests that the participants consider that a complex system directly implies a certain complexity of the tasks carried out on that system and vice versa.

Finally, we also asked the participants to specify the availability of documentation (beyond the source code) of the VSS. An even distribution was found between the three categories "almost complete documentation", "incomplete documentation", and "little or no documentation at all".

3.3. Functional aspects of SV tools

In an effort to rate the usefulness of functional aspects and features related to SV tools, we presented the participants 34 aspects, together with four possible answers for each of them ("absolutely essential", "useful, but not essential", "not at all useful", and "don't know/not applicable"). Table 1 depicts the list of aspects.

We classified these aspects using two different methods:

1) By identifying the aspects that were selected as "absolutely essential" by at least 50% of the participants.
2) By assigning a weight to the possible answers (2 for "absolutely essential", 1 for "useful, but not essential", and 0 for "not at all useful"), by calculating the mean of all the participants' answers for each one of the 34 aspects, and by selecting the aspects for which the average is higher than 1.5.

The usefulness of the functional aspects will be first discussed independently from any factors of influence (Section 3.3.1), and then by taking into account some influencing factors (Section 3.3.2). These factors relate to the participants and to the VSS, and were identified by

Table 1. Functional aspects of SV tools.

#	Functional aspects
F1	Source code visualization (textual views)
F2	Source code browsing
F3	Direct access (without passing via high-level objects) to the source code
F4	Symbol list visualization (e.g., the files of the visualized software, the defined functions and the data types (variables, constants, etc.))
F5	Easy access, from the symbol list, to the corresponding source code
F6	Graph visualization
F7	Synchronized browsing of views of the same type or of a different type (e.g., synchronized graphic and textual browsing)
F8	Navigating from one view to another one of the same or of a different type (cross-referencing; e.g., navigating from source code to a graphic window and vice-versa)
F9	Possibility of having multiple views of the same type or of different types, with instances of the same object being highlighted in all the views (e.g., possibility of relating the nodes and arcs in graphs to the corresponding source code)
F10	Automatic layout capabilities
F11	Automatic scaling capabilities
F12	Abstraction mechanisms (e.g., display of subsystem nodes, and of composite arcs in a graph)
F13	Overview windows (e.g., show the hierarchical structure of the visualized software)
F14	Search tools for graphical and/or textual elements
F15	Filtering capabilities
F16	Zoom-in and zoom-out capabilities
F17	Dynamic visualization (visualization of the software at runtime)
F18	Hierarchical representations (of subsystems, classes, etc.)
F19	Navigation across hierarchies (of subsystems, classes, etc.)
F20	Hypertext style navigation
F21	Arbitrary navigation
F22	Possibility to record the steps of an arbitrary navigation
F23	Possibility to return back to preceding navigation steps
F24	Saving of views for future use
F25	Possibility to display or to hide information on demand
F26	Program slicing
F27	Possibility to view the current focus (e.g., fish-eye views, detailed views of interesting nodes, highlighting nodes and arcs, thumbnail images of source code, etc.)
F28	Possibility to view the path which leads to the current focus (histories or breadcrumb trails)
F29	Possibility to customize the visualization
F30	Possibility to provide summary comprehension (coarse-grained)
F31	Possibility to provide detailed comprehension (fine-grained)
F32	Use of colors
F33	Animation effects
F34	3D representations and layouts, and virtual reality techniques

considering the various exclusive samples of the survey:

- Samples related to different facets of the participants:
  - Work environment ("company" versus "research setting")
  - Size of workgroup ("small" versus "large")
  - Knowledge level of VSS ("low" versus "high")
- Samples related to different characteristics of the VSS:
  - Implementation language ("object-oriented" versus "procedural" versus "declarative")
  - Size ("small" versus "large")
  - Level of complexity ("simple" versus "complex")

Finally, some interesting relationships between the various functional aspects will also be presented (Section 3.3.3).

3.3.1. Usefulness of functional aspects. Though differing slightly in their respective order of appearance, the same seven functional aspects were identified when applying the one or the other of the two methods mentioned above. Applying the second method and presenting the aspects in decreasing order yields the following list: F14 (1.72, 74%), F1 (1.70, 69%), F18 (1.61, 57%), F32 (1.60, 63%), F2 (1.55, 56%), F19 (1.54, 57%), F5 (1.52, 50%) (see Figures 2 and 1).

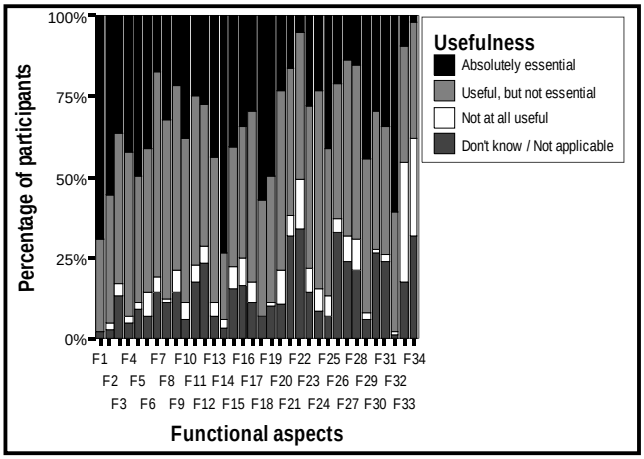

Figure 1. Usefulness of functional aspects (represented by percentages).

Ten aspects were identified as "useful, but not essential" by more than 50% of the participants: F7 (62%), F24 (58%), F9 (55%), F8 (55%), F27 (53%), F28 (53%), F20 (53%), F17 (52%), F11 (51%), and F4 (51%). A mean higher than 1 (and lower than 1.5) was found for each of these aspects. However, most of them are situated at the low end of those 24 aspects that have a mean between 1 and 1.5.

Three aspects, finally, obtained a mean of usefulness lower than 1: F22, F33, and F34 (see Figure 2).

Some participants opted for "don't know/not applicable" for certain functional aspects. Two possible explanations come to mind: the participant (1) did not understand or did not have an opinion about the aspect, or (2) the aspect did not apply in his or her context. The latter explanation would be plausible, for instance, for aspect F3 ("direct access to the source code"), in case the participant's main tasks are at the analysis and design level, using a tool such as Together/J [37] or Rational Rose [19]. Regarding the first explanation, it is possible that aspects such as F21, F22, F26, F30, or F31 were not enough comprehensible for all participants in spite of the explanations that were provided as footnotes in the questionnaire. The aspect F34,

finally, is quite an uncommon aspect in current SV tools, thus suggesting a high degree of incomprehensibility (see Figure 1).

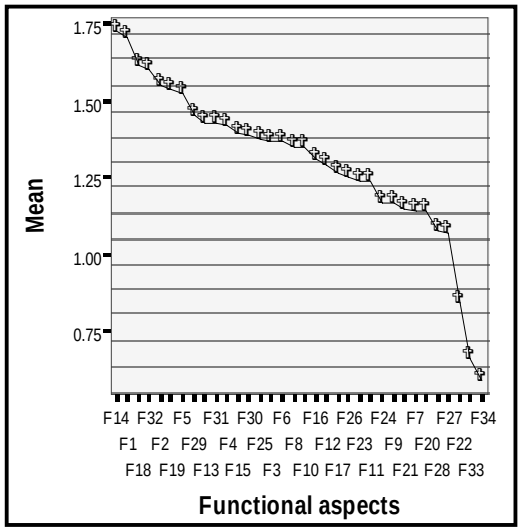

Figure 2. Usefulness of functional aspects (represented by means in decreasing order).

3.3.2. Factors of influence on usefulness of functional aspects. In this section, the various exclusive samples of the survey, as explained above in the introduction of Section 3.3, are examined in respect to the usefulness of functional aspects. We first discuss the three samples relating to the participants, and then the three samples concerning the VSS.

We noticed that participants from research settings attached more importance to the two aspects F33 and F34 than participants from industry (F33: mean of the answers is 0.97 versus 0.49, t-test value = -3.113, $p < 0.003$; F34: 0.77 versus 0.48, t-test value = -2.202, $p < 0.032$). As we can see, the t-test suggests to reject the null hypothesis (H_0) which is the equality of the two means.

A positive correlation (though weak) was identified between F6 and the size of the workgroup (tau-c = 0.206). A possible explanation could be that graph visualization provides in general a coarse-grained view of the VSS (or part of it), and that large workgroups tend to be involved in large projects where coarse-grained views are essential for getting the big picture of the VSS.

The weaker the level of knowledge of the VSS is, the more the participant needed a global view of the system before manipulating its source code, that is, the higher was the importance attached to F30 (tau-c = -0.156). On the other hand, the more the level of knowledge of the VSS was high, the higher was the importance attached to F31 and F1 (tau-c = 0.200).

Turning now to the samples pertaining the characteristics of the VSS, the percentage of participants who chose F1 (respectively, F14) as an "absolutely essential" aspect raises to 82% (respectively, 82%) when the VSS were procedural, and to 69% (respectively, 79%) in the case of object-oriented VSS, suggesting that the procedural world attaches more importance to visualization of the source code than the object-oriented world. On the other hand, higher percentages were granted to the two aspects F18 and F19 in the case of object-oriented VSS. In the case of declarative VSS, F17 and F33 were identified as the most essential aspects by most participants (4 out of 5).

Participants dealing with large VSS attached more importance to a higher number of functional aspects than those dealing with small VSS. Moreover, the choice "don't know/not applicable" was more often selected in the case of small VSS than in the case of large VSS. A positive correlation was identified between F6 and the size of the VSS (tau-c = 0.239), whereas a negative correlation was identified between F3 and the size of the VSS (tau-c = -2.282). This suggests that the more the VSS is large, the more important it is to visualize it graphically and the less useful it is to pass directly to the source code. When participants work with large VSS rather than with small VSS, they attach more importance to the four aspects F4, F5, F6, and F24 (for all these aspects, the t-test suggested to reject H_0; for details refer to [2]).

We noticed that when the VSS were of low complexity, the participants attached significantly less importance to most of the functional aspects than when they were complex, and the answer "don't know/not applicable" was largely selected. Furthermore, positive correlations higher than 0.300 were identified between the complexity level of VSS and the three aspects F6 (tau-c = 0.321), F16 (tau-c = 0.320), and F19 (tau-c = 0.314). A great difference between the means was also detected for these three aspects (the t-test suggests to reject H_0).

3.3.3. Relationships between functional aspects. We detected a number of correlations that simply confirm the coherence of the collected data. Some examples thereof are F4 and F5 (tau-b = 0.579), F4 and F15 (tau-b = 0.214), F21 and F22 (tau-b = 0.386), and F27 and F28 (tau-b = 0.417).

Other correlations suggest more interesting interdependencies. For example, significant positive correlations exist between F6 and F7 (tau-b = 0.300), F6 and F8 (tau-b = 0.302), and F7 and F8 (tau-b = 0.406). This suggests that participants are "code-centric", that is, they wish to have at the same time graphic visualization as well as access to the source code. Another interesting correlation exists between F14 and F21 (tau-b = 0.301), indicating that participants realize the need for search tools for arbitrary navigation. The two aspects F17 and F33 (tau-b = 0.379) are to some extent dependent; indeed, dynamic visualization is often carried out by using animation techniques. The kinds of

comprehension proposed by the two aspects F30 and F31 (tau-b = 0.599) are complementary. Finally, F33 and F34 (tau-b = 0.505) are similar, in that both are related to extended capabilities of SV tools.

3.4. Practical aspects of SV tools

To assess a number of practical aspects of SV tools, we presented the participants 13 aspects, together with four possible answers for each of them ("very important", "fairly important", "not important", and "don't know"). Table 2 presents the list of the 13 aspects.

Table 2. Practical aspects of SV tools.

#	Practical aspects
P1	Cost of the tool
P2	Content and quality of the documentation
P3	Availability of on-line documentation
P4	Availability of technical support
P5	Tool performance (execution speed of the tool)
P6	Tool reliability (absence of bugs)
P7	Ease of learning and installation of the tool
P8	Ease of using the tool (e.g., no cumbersome functionality)
P9	Ease of visualizing large-scale software
P10	Visualization of software written in different programming languages
P11	Portability of the tool
P12	Quality of user interface (intuitive widgets)
P13	Possibility of transforming visualization output into different file formats (postscript, pdf, bmp, etc.)

For classifying the practical aspects, we applied the same methods as for the functional aspects: 1) calculation of frequencies in percentages, and 2) calculation of means (see Section 3.3).

The importance of these factors will be first discussed independently of any factor of influence (Section 3.4.1). Then, in Section 3.4.2, we will take into account two factors of influence, namely, work environment of the participant, and size of the VSS. Note that the other four factors specified in Section 3.3 did not impact the importance ratings of the practical aspects.

3.4.1. Importance of practical aspects. Among the 13 practical aspects, six were at the same time selected as being very important by more than 50% of the participants, and classified with a mean of importance higher than 1.5. Independently of the method, we obtained the following ordering: P6 (88%, 1.87), P8 (72%, 1.70), P12 (69%, 1.68), P9 (66%, 1.64), P3 (63%, 1.58) and P2 (62%, 1.56). No aspect was classified as being not important; in fact, the mean of importance for each aspect is above 1.

Beyond rating the 13 practical aspects mentioned above, participants were invited to come up with additional aspects they did consider important. Five principal patterns of answers were detected:

- Integrate into the SV tools specific tools other than code generators (design partitioning, metrics, etc.) (10 participants)
- Integrate tools for code generation (5 participants)
- Be able to import from/export to other SV tool formats (5 participants)
- Extend the SV tool with a macro-like language (2 participants)
- Integrate SV tools into development environments and allow for round-trip engineering (2 participants)

A few other aspects were proposed as well; yet it turned out that most of them were simply taken and reformulated from the section on functional aspects and from the list of the 13 practical aspects. A cross-check with the answers already given for those aspects did not reveal any inconsistency.

3.4.2. Factors of influence on importance of practical aspects. In terms of work environment, we noticed that participants from research settings attached considerably more importance to the two aspects P1 and P4 than those from industry. These results were initially generated according to the frequency of the answer "very important" (for P1, 50% versus 32%; for P4, 50% versus 25%), and subsequently confirmed by calculating the mean of the respective answers. Furthermore, we observed a remarkable difference between the two means calculated for P10, and this in favor of research settings (1.44 versus 1.10, t-test value = -2.304, $p < 0.024$; once again, the t-test makes it possible to reject H_0). Referring to Figure 3, the black line joining the means for research settings ("academia") is in general located above the gray line which joins the means for industry, except for the two aspects P5 and P9. However, the t-test calculated for these two aspects does not suggest to reject H_0.

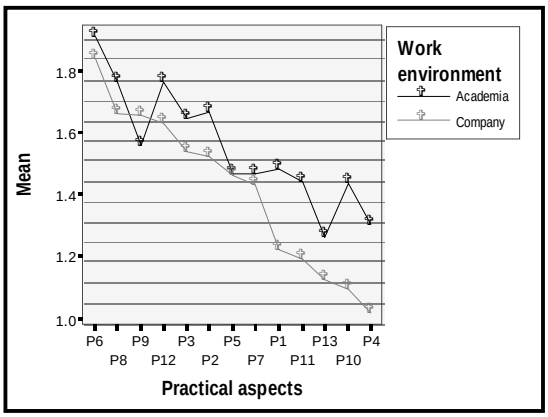

Figure 3. Importance of practical aspects of SV tools considering the work environment (represented by means; company means in decreasing order).

Considering the size of the VSS, we noticed that participants dealing with large systems attached more importance to the majority of the practical aspects than those dealing with small ones. The most obvious aspect for which we can reject H_0 is P9 (1.75 versus 1.39, t-test value = -2.268, $p < 0.028$). Indeed, the more the developers deal with large systems, the more they feel a need for a SV tool that facilitates the visualization of such systems.

3.5. Cognitive evaluation of SV tools

Recall that for tackling the section related to cognitive aspects (as well as the section on benefits and improvements, and Part II), the participants were asked to choose a specific tool for which they would answer the questions ("SV tool at hand"). We received replies for more than 40 different SV tools. Given the grand total of 107 participants, this led to a small sample size for each tool, and consequently, the results for a specific tool have little significance.[3] However, we obtained a number of interesting global results, as exposed below.

More than 50% of the participants specified that the SV tool at hand is very easy to use (63%), that it is able to generate useful results (68%), and that they have confidence in the generated results (64%). However, a notable gap was identified between some of the desired practical aspects (Table 2) and the practicality of the SV tool at hand. Specifically, 69% of the participants rated the aspect P12 as "very important", yet only 43% answered a similar question in the cognitive aspects section with "high". A similar gap was found for the aspect P8 (72% for "very important", and only 63% for "high").

Furthermore, the results confirm our intuition that strong support for code comprehension facilitates tasks such as maintenance, debugging, testing, adding new functionalities, improving documentation, and analyzing code. In fact, positive correlations were found between SV tool support for each one of these tasks and for the code comprehension activity, with the most significant one being the correlation between analyzing code and code comprehension (tau-b = 0.515).

The participants were also asked to assess the extent to which certain characteristics of the VSS decrease the effectiveness of the SV tool at hand. We noticed that "insufficient documentation" is the one which decreases the most the effectiveness of a SV tool (57% of the participants answered "a lot/more or less"), followed by the "code complexity" characteristic (50%). On the other hand, the participants seem to be satisfied with the scalability of the SV tool at hand (55% of the participants answered the question concerning "size of code" with "not at all"). Note that the participants came up with some additional characteristics impeding the effectiveness of the tool at hand. However, these characteristics turn out to be specific to particular tools, such as Fujaba [7], UML Studio [32], and Rational Rose [19].

Finally, the majority (74%) of the participants found that the SV tool at hand affects to some extent the approach taken to carry out a given task on the VSS. Also, a majority (70%) expressed an excellent degree of satisfaction concerning the results obtained from their respective SV tool, and none of the participants were completely unsatisfied with the SV tool they were using.

3.6. Benefits and improvements of SV tools

The participants were asked two open-ended questions concerning (1) the benefits drawn from using the SV tool at hand, and (2) the principal improvements that should be done to that tool.

The patterns that we detected for the benefits and which concerned multiple tools are the following:

- Savings in time when carrying out certain tasks on the VSS (15 SV tools)
- Better comprehension (6 SV tools)
- Increase in productivity (speed of development, etc.) (4 SV tools)
- Management of complexity (4 SV tools)
- Assistance to find errors in the code (3 SV tools)
- Improvement of quality (code, processes, etc.) (3 SV tools)
- Saving money (2 SV tools)

A few benefits were only mentioned for single tools. In addition, some general opinions about SV, not related to a particular SV tool, were expressed, including "in general, SV tools make software engineering an easy and enjoyable task."

In respect to improvements, many aspects that were already stated in earlier sections of the questionnaire were brought up, together with various additional aspects (which we compiled and presented as patterns in Section 3.4.1). Furthermore, several new aspects were suggested, including:

- Provide help system (5 SV tools)
- Adapt the SV tool to compilation (4 SV tools)
- Add more functionality in the tool (3 SV tools)

[3] Note also that the participants were assured that the supplied information would be kept strictly confidential.

4. Survey results of Part II

4.1. Technical information about SV tools

Part II of the questionnaire was filled out by 41 participants for an overall of 24 different SV tools. 16 of the tools are commercial, whereas eight are research prototypes. Furthermore, we noticed that 12 tools are made of only one large tool, whereas the other 12 tools are made of several tools.

One of the questions asked about the purpose for which the SV tool at hand can be used. The three most frequent answers were: software comprehension (18 tools), analysis (17), and addition of new functionalities (16); the least quoted purpose was "design" with only four tools (Rational Rose [19], Rhapsody [20], Together/J [37], and UML Studio [32]). A total of ten participants benefited from the "other" option to specify other purposes for which their tool can be used. The participant of The Paislei IDE [35], for instance, mentioned that the tool is used for the design and visualization of grammars. The participants of aiSee [1] specified the two purposes "selling complex systems" and "teaching". The participants of Clarion [4], Grasp [10] and UltraEdit [39] added "creation of code, development and writing" as a purpose. This latter purpose, though not specified, is of course also supported by other tools, such as Emacs [8]. Finally, the "reverse engineering" purpose was added several times (UML Studio [32], Fujaba [7], PBS [17]), and again, is also supported by further tools of the survey, such as SHriMP [25] and Rigi [21].

16 out of the 24 SV tools were considered full-blown software development environments (or part of an environment). We notice that in Part I, this aspect was largely wished by the participants. Consequently, we are optimistic about the result we got here.

The last question of the section on technical SV tool information required to specify the programming language(s) in which the VSS must be written. We noticed that the majority (17) of the tools accept systems written in different languages. Note that, somewhat surprisingly, the practical aspect P10 corresponding to "visualization of software written in different programming languages" was not classified among the six most important practical aspects (cf. Section 3.4.1). The seven remaining tools either accepted systems written in only one language (e.g., C++, Java, or Pascal), or they required an adapted parser to create files in a specific format (e.g., the *graph description language* for VCG [40], or the *term representation* for daVinci [34]). We realized that a strong majority of the SV tools visualize object-oriented systems (Java and C++); in fact, 20 out of the 24 SV tools support one or the other of these languages, or both.

4.2. Support of code analysis by SV tools

In the second section of Part II, we were interested in the SV tools' capabilities for code analysis of the VSS. The questionnaire covers the seven main aspects: (1) function calls, (2) function cloning, (3) inheritance graphs, (4) subsystem architecture graphs, (5) different levels of detail, (6) metrics, and (7) multiple versions. In turn, the main aspects are refined into 24 subaspects which are presented as questions to the participants. Each question has the four mutually exclusive answers: "yes" ("y", the SV tool at hand provides the aspect), "no" ("n"), "don't know/not applicable" ("?"), and "no/don't know, but I would like this feature" ("!").

Note that within the main aspects, certain dependencies exist, that is, given one main aspect, a "no" answer to one of its subaspects may exclude "yes" answers to other subaspects of that main aspect. In the on-line version of the questionnaire, these dependencies are being enforced by appropriate JavaScript code.

Below, we first explain the processing of multiple replies (Section 4.2.1). Then, in Section 4.2.2, we discuss some code analysis results concerning the set of 24 tools surveyed.

4.2.1. Processing of multiple replies. For six tools, out of the 24 different tools covered by the 41 participants, we obtained multiple replies as follows (number of participants in parentheses):

- Grasp [10] (6)
- pcGrasp [10] (5)
- aiSee [1] (5)
- PBS [17] (3)
- Rational Rose [19] (2)
- Software through Pictures [27] (2)

Not surprisingly, for each of these tools, we came across some conflicting replies when examining the different subaspects. In order to obtain for each subaspect and each tool one single reply, we proceeded in two steps.

As a first step, we defined heuristics for mapping a set of conflicting replies into one single reply, as follows ("#" means the number of elements, and "yn" specifies that the aspect is more or less supported by the tool):

(1) $(\#y = 1 \wedge \#n = 1) \vee (\#y > 1 \wedge \#n > 1) \Rightarrow$ yn
(2) $[(\#y \leq 1 \wedge \#n > 1) \vee (\#y = 0 \wedge \#n = 1)] \wedge \#! = 0 \Rightarrow$ n
(3) $(\#y > 1 \wedge \#n \leq 1) \vee (\#y = 1 \wedge \#n = 0) \Rightarrow$ y
(4) $[(\#y < \#n) \vee (\#y = 0 \wedge \#n = 0)] \wedge \#! \geq 1 \Rightarrow$!
(5) $\#y = 0 \wedge \#n = 0 \wedge \#? \geq 1 \wedge \#! = 0 \Rightarrow$?

After executing a script implementing these heuristics, we ended up with four "yn" replies, with two of them violating the dependencies mentioned above in the introduction of Section 4.2. Therefore, as a second step, we tried

and eliminated all "yn" replies, by close examination of the subaspect and the documentation of the SV tool at stake, and in one case, by consulting a trusted participant. (For further details on the procedure, refer to [3].)

4.2.2. Results for code analysis support. Among the seven main aspects, visualization of function calls is the one being most supported, that is, by 18 tools. The next aspect is visualization of inheritance graphs with 17 tools, followed by visualization of different levels of detail in separate windows with 15 tools. The remaining four main aspects are supported by fewer than 12 tools. Note that within one main aspect, highly specialized subaspects were less supported. This was also the case for the three top main aspects mentioned above.

The two main aspects supported by the smallest number of tools were the detection of function cloning with 6 tools, and calculation of metrics with 5 tools. Though not supported, these aspects were seldom wished. Indeed, whereas 10 tools do not support the detection of function cloning, we have only 3 tools for which this feature was wished. For the calculation of metrics we obtained similar results, the respective numbers being 8 and 3. We also realized that a notable number of participants answered "?" for these two aspects. In fact, the three least known aspects are: calculation of metrics with 7 tools, detection of function cloning with 4 tools, and visualization of subsystem architecture graphs with 4 tools as well.

Our results suggest that users of specific SV tools often wish to have refinements of the aspects that are already supported:[4] if an aspect does not exist in a tool, the user of that tool seldom feels the need to have it; on the other hand, if it exists, he or she would like to have some of the related subaspects. Thus, it is not surprising that the subaspects related to the two top main aspects (visualization of function calls and visualization of inheritance graphs) were placed among the ones the most wished. Here are, in a decreasing order, these subaspects, preceded by the corresponding main aspect:

- Function calls: Present complex recursion chains as a whole in the same graph (5 tools do not support this subaspect, and for 6 tools this subaspect was wished)
- Inheritance graphs: Make it possible to detect deep inheritance branches (1 versus 6)
- Subsystem architecture graphs: Provide a way to relate the subsystem architecture graph to other results of the SV tool (e.g., areas of different colors in the graph) (3 versus 5)
- Inheritance graphs: Make it possible to detect occurrences of multiple inheritance (1 versus 4)
- Function calls: Represent the functions of recursion chains by nodes and give the possibility to reach directly the source code from the nodes (4 versus 4)

Among the least supported subaspects were the elimination of simple recursive functions (only supported by The Paislei IDE tool [35]), and the reduction of the granularity during the detection of the differences which exist in the clones of a function (only supported by Rhapsody [20]). Finally, none of the 24 tools makes it possible to integrate metrics and source code.

Among the 24 tools were two text editors, Emacs [8] and UltraEdit [39]. Naturally, the majority of the subaspects related to code analysis are not supported by these tools.

5. Discussion and perspectives

5.1. Lessons learned

As with most questionnaires, there is room for improvement. For example, in a future version of the questionnaire, we would like to present finer grained choices for some of the multiple-choice questions. Furthermore, since we obtained a lot of interesting information from the open-ended questions, we would probably put even more emphasis on that type of question. For asking about the usefulness of functional aspects, for instance, open-ended questions could yield complementary data. Yet another improvement would be to distribute different versions of the questionnaire for different kinds of SV tools (design phase, implementation phase, etc.), to obtain more targeted results.

Concerning the cognitive evaluation of SV tools, we realize that, in order to generate tool-specific results of higher significance, it would have been necessary to have more participants per tool than we obtained in the current survey. On the other hand, just one single participant per tool would have probably been enough to conduct Part II of the survey, providing, of course, that these individuals were true experts of the tool at hand.

5.2. Summary and impact of findings

In general, the participants were quite pleased with the SV tool at hand. Functional aspects such as searching and browsing, use of colors, and easy access from the symbol list to the corresponding source code were rated as the most essential aspects. Hierarchical representations as well as navigation across hierarchies were also strongly desired, especially when the VSS was implemented using an object-oriented language. On the other hand, source code visualization was almost always wished when the VSS was procedural. The three aspects that were least appreciated, especially in industry, were animation effects, 3D visuali-

[4] *L'appétit vient en mangeant*, as the French proverb goes – appetite comes with eating.

zation and VR techniques. However, animation was identified as being quite useful when the VSS was implemented in a declarative language.

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. Many participants also rated the quality of the user interface as highly important. 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, in terms of content, quality, and on-line availability. 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.

Both functional and practical aspects were influenced by the two factors work environment and size of the VSS. Yet, the usefulness of functional aspects was also influenced by the size of workgroup and the knowledge level of the VSS, as well as by the implementation language and the level of complexity of the VSS. Recall that these latter factors did not have any measurable impact on the importance of the practical aspects.

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 the 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. In general, the users are satisfied with the results obtained from applying SV tools. They specified many benefits from using the SV tool at hand. The two most quoted ones were the savings in time when carrying out a specific task, and better comprehension of the VSS. However, numerous desired aspects and improvements to the SV tool at hand 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 (1) integrate third-party tools, (2) make it possible to import from/export to other SV tool formats, and (3) are more customizable in terms of compiler settings and hence more adapted to the compiler system(s) used for code development. A more complete list of the improvements that should be done to SV tools was presented in Sections 3.4.1 and 3.6. These items not only concern tool builders, but some of them constitute issues of a research agenda.

5.3. Future work

There are three directions in which we plan to continue this research.

For one thing, most of the analysis reported in this paper is descriptive (frequency and means calculation), with some correlation and t-test analysis. As a next step, we plan to perform a stronger and more meaningful analysis using factor analysis to find the principal factors that emerge from the participants' pattern of answers to the question set. This could be followed by cluster analysis, which permits grouping the participants by their patterns of responses in a two-dimensional space. Using these techniques may lead to firmer conclusions, allowing us to determine for instance, which participants answered in a similar way.

As a second direction, via multi-phase survey techniques [5], we aim to identify representative target populations for different survey questions. We will attempt to test significant support of hypotheses with more advanced statistical techniques than the ones already used for analyzing the data collected with the current questionnaire.

As a third direction of future research, we want to carry over metaphors, techniques, and tools found and empirically evaluated during the different phases of the survey, 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.

Acknowledgement

First and foremost, we would like to thank all 107 participants for filling out the questionnaire. We are also grateful to the people who helped us in spreading the URL of the questionnaire all around the world. Finally, our thanks go to the colleagues in our group and in CSER, in particular to Janice Singer from NRC Ottawa, who helped improve the early versions of the questionnaire.

References

[1] aiSee (AbsInt Angewandte Informatik GmbH). On-line at <http://www.absint.de/aisee.html>.

[2] Bassil, S. and Keller, R. K., Analysis of the Questionnaire on Software Visualization Tools – Part I. *Technical Report GELO-132*, Université de Montréal, September 2000. In French.

[3] Bassil, S. and Keller, R. K., Analysis of the Questionnaire on Software Visualization Tools – Part II. *Technical Report GELO-133*, Université de Montréal, September 2000. In French.

[4] Clarion by SoftVelocity. On-line at <http://www.softvelocity.com/products/products.htm>.

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

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

[7] Fujaba Homepage. On-line at <http://www.uni-paderborn.de/fachbereich/AG/schaefer/ag_dt/PG/Fujaba/>.

[8] GNU Emacs (Free Software Foundation) <http://www.gnu.org/software/emacs/emacs.html>.

[9] GraphViz (AT&T Labs-Research). On-line at <http://www.research.att.com/sw/tools/graphviz/>.

[10] Grasp: Graphical Representations of Algorithms, Structures, and Processes. On-line at <http://www.eng.auburn.edu/department/cse/research/grasp/>.

[11] Harel, D., Biting the Silver Bullet. Toward a Brighter Future for System Development. *IEEE Computer*, 25(1), pages 8-20, January 1992.

[12] 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)*, pages 226-235, Los Angeles, CA, USA, May 1999.

[13] Keller, R. K., User Interface Tools: a Survey and Perspective. In *R. N. Taylor and J. Coutaz, editors, Software Engineering and Human-Computer Interaction*, pages 225-231, 1995. Springer. LNCS 896.

[14] Lethbridge, T. C., What Knowledge is Important to a Software Professional? *IEEE Computer*, 33(5), pages 44-50, May 2000.

[15] Myers, B. A. and Rosson, M. B., Survey on User Interface Programming. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'92)*, pages 195-202, Monterey, CA, USA, May 1992.

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

[17] PBS (Portable Bookshelf). On-line at <http://www.turing.toronto.edu/pbs/>.

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

[19] Rational Rose. On-line at <http://www.rational.com/products/rose/index.jsp>.

[20] Rhapsody (I-Logix). On-line at <http://www.ilogix.com/>.

[21] Rigi: A Visual Tool For Understanding Legacy Systems. On-line at <http://www.rigi.csc.uvic.ca/>.

[22] 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)*, pages 22-32, San Jose, CA, USA, October 2000.

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

[24] Schauer, R. and Keller, R. K., Pattern Visualization for Software Comprehension. In *6th International Workshop on Program Comprehension (IWPC'98)*, pages 153-160, Ischia, Italy, June 1998.

[25] SHriMPage. On-line at <http://www.csc.uvic.ca/~wjw/shrimpage.html>.

[26] SNiFF+ (TakeFive). On-line at <http://www.takefive.com/products/sniff+.html>.

[27] Software through Pictures (Aonix). On-line at <http://www.aonix.com/Products/SMS/softbench_integ.html>.

[28] SPSS Statistical Package for the Social Sciences. On-line at <http://www.spss.com/>.

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

[30] 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, pages 171-185, January 1999.

[31] Storey, M.-A. D., Wong, K., and Müller, H. A., How Do Program Understanding Tools Affect How Programmers Understand Programs? In *Proceedings of the 4th Working Conference on Reverse Engineering (WCRE'97)*, pages 12-21, Amsterdam, Holland, October 1997.

[32] Stringray UML Studio Product Page. On-line at <http://www.stingray.com/products/umlstudio/>.

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

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

[35] The Paislei IDE and Laleh's Pattern Matcher. On-line at <http://qtj.dhs.org/~lpm/>.

[36] TkSee (Knowledge-Based Reverse Engineering). On-line at <http://www.site.uottawa.ca/~tcl/kbre/>.

[37] TogetherSoft: Together/J. On-line at <http://www.togethersoft.com/together/togetherJ.html>.

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

[39] UltraEdit (IDM Computer, Solutions, Inc.). On-line at <http://www.idmcomp.com/products/index.html>.

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

[41] Watson, A. B. and Buchanan, J. T., Towards Supporting Software Maintenance with Visualisation Techniques. *Technical Report R5*, University of Strathclyde, 1995.