The Internet Software Visualization Laboratory1

ABSTRACT

The Internet Software Visualization Laboratory (ISVL) combines our research in Software Visualization and teaching over the internet to tackle the problem of how computer programming can be effectively taught to students working from home. Our approach provides a rich, collaborative environment for exploring and demonstrating programming constructs. ISVL supports both asynchronous and synchronous working, and allows students to move seamlessly from a tutor-led teaching scenario, to personal or peer exploration. ISVL provides a rich source of empirical data which will shed light on how Software Visualization can be most effectively incorporated within a computer programming curriculum.

INTRODUCTION

Teaching computer computer programming has long been known to be a complex process. This problem is compounded on our own courses where teaching is carried out a distance, raising the following issues:

1) The lack of face-to-face support - contact between student and tutor is either by telephone or email, meaning the tutor establishing the nature of the students? problem can take a great deal of time and be prone to misunderstanding.

2) Students can rarely work together - as the students work part time and fit their studies around other committments it is impossible for them to work together synchronously.

3) The lack of a computer laboratory - students work at home using a range of computers, many of fairly low specification, meaning a range of platforms have to be supported.

The approach taken to alleviate these problems was to develop a software visualization laboratory. Software Visualization (SV) is the use of graphical and textual formalisms to describe the execution of computer programs. We view SV as having a valuable role within the teaching of computer programming, particularly for languages with a complex execution model. Our view is influenced by empirical work which found that a crucial objective when teaching programming was to provide a clear execution model (Eisenstadt et al. 1985). This fits

well with the aims behind SV. A key role envisaged for SV is in
computer science education. SV permits students to have an overall
conceptual story of how the program works, explore and debug programs,
and use the visualizations as a form of language in which to convey
their understanding. The laboratory has the folowing characteristics:

1) an internet client server architecture using platform independent
software for the client (a java compliant browser) and placing most of
the work on the server (to support low specification machines),

2) an environment that can provide a context for synchronous
communication, so that the tutor can quickly establish what the
student wishes to discuss, and

3) a facility which enables asynchronous communication by allowing
tutor and student explanations to be archived for future reference.

The approach of using SV over a network as an educational tool raises
futher issues as to how SV can be most appropriately included within a
computer programming curriculum. The key theoretical issue we to
address concerns the use of single or multiple representations to
support learning. One one hand specific representations could be
hand-picked to help student at particular stages of the course (i.e. a

1Submitted to PPIG?97, Psychology of Programming Interest Group 9th
Annual Workshop Jan. 3rd - 5th 1997 Sheffield Hallam University,
Sheffield, UK

stage appropriate approach). This means each representation will be
carefully tailored to the students needs at that time, but will mean
the student will have to learn a greater number of representations. On
the other hand a single ?best bet? representation could be used
throughout (i.e. a cradle-to-grave approach). Our new laboratory will
allow us to investigate this issue.

The next section describes the theoretical underpinnings of our
approach. This is followed by an outline of the design requirements
for the new environment and a tour of the ISVL environment. The final
section contains a brief discussion on the benefits of ISVL and some
conclusions.

THEORETICAL BASIS

Research into SV was inspired by the Sorting Out Sorting film (Baecker and Sherman, 1981) which used animations to explain sorting algorithms. SVs can be used to present different perspectives of program execution, and are generally classified into two distinct groups: Algorithm Animation systems and Program Visualization systems. Algorithm Animation systems illustrate program execution in an algorithmic language independent way. Notable examples include BALSA (Brown, 1987), Zeus (Brown, 1991) and AACE (Gloor, 1992). Program Visualizations such as the Transparent Prolog Machine (Eisenstadt and Brayshaw, 1988; Brayshaw and Eisenstadt, 1991) for Prolog and ZStep for Lisp (Lieberman and Fry, 1995) present an account of execution at the code level. ISVL takes the form of a visualization framework, able to represent execution on both the algorithmic and code level. Such frameworks already exist, including TANGO (Stasko, 1990) and Viz (Domingue et al, 1992), though ISVL is a platform independent internet-based environment, supporting other aspects of the educational experience including peer collaboration, tutorials and the handling of assignments. ISVL will form a central component of a new web-based programming course.

Though many claims have been made by SV designers for the educational suitability of their product, little has been said as to the stage in the learning process when particular visualizations are most appropriate. Conversely, some SV designers have claimed their product applies equally well from novice through to expert. For example, this claim was made by Brayshaw and Eisenstadt (1991) for the Transparent Prolog Machine(TPM). TPM presents program execution graphically as an AND/OR tree, in order to give the detailed account of execution an ?at a glance? quality allowing information to accessed quickly (see figure 1).

Figure 1. The Transparent Prolog Machine (TPM) AND/OR tree showing an overall perspective of the execution, and a TPM fine-grained view of the top most node showing execution details.

An alternative viewpoint would be that particular types of representation are appropriate for particular stages in the learning process. This debate can be summarised as a cradle to grave versus stage appropriate view of how SV should be integrated into the learning experience. There is a wealth of evidence to suggest that the kind of knowledge structures and strategies used by expert and novice computer programmers differ qualitatively rather than quantitatively, and this affects how they are able to use different kinds of SV. For example, Chi et al. (1981) found that novices tend to focus on the surface features of the language whereas experts tend to use higher level structural representations.

Recent SV evaluation studies suggest important differences in the way novices and experts are able use SVs (Mulholland, 1995). The empirical work suggests that novices tend to utilise a fairly specific group of strategies which use core programming information. These were termed comprehension strategies. Three types of comprehension strategy were deemed central to the effective novice use of the SV. These were review, mapping and test strategies.

A review strategy was defined as using the historical perspective present in the SV display to work out how the current position in the execution was reached. A test strategy was defined as predicting how the execution will develop during future steps of the SV and then moving forward to test this prediction. A mapping strategy was the process of mapping between the information contained in the SV and the source code being visualized. Novices were only able to use these strategies if the information on which they relied was explicit. Implicit information, using what Petre and Green (1990) term the secondary notation, could not be utilised effectively by the novices. These findings motivated the development of the Theseus SV for Prolog, which has been found to be very effective for a novice population (see figure 2).

Experts, on the other hand, will rely on explicit information when initially unfamiliar with the display but unlike novices are later able to use the secondary notation to draw out high level patterns (Mulholland, 1995). This indicates that experts, when using tools such as SVs are able to draw on a qualitatively distinct set of strategies. This leads to an interesting dilemma when considering how SV should be incorporated into a computer programming curriculum. As the student commences the course as a novice and (hopefully) completes the course as an expert; what kind(s) of SV should be used? On one hand the novice could start by using a ?novice SV? (e.g. Theseus) and be introduced to an ?expert SV? (e.g. TPM) at a later time, i.e. a stage appropriate approach. This has the advantage of using SVs appropriate to each learning stage, though the disadvantage of students being required to shift from one supporting representation to another at some intermediate stage. On the other hand, the novice could be initially introduced to the ?expert SV? and helped to grow into it, i.e. a cradle to grave approach. This has the advantage of having only one supporting representation to learn during the course. It has the disadvantage of the students requiring extra support from day one in the use of a complex environment.

This conundrum lead us to the development of the ISVL environment, whereby a range of SVs can be used under one roof, and different kinds of support can be provided to help the students integrate the SV into the wider learning experience. Support is provided by allowing the SV to be used collaboratively as well as individually within the same environment. This draws on the work of Chi et al. (1989) who have shown the beneficial effects of self explanation on learning. ISVL therefore redefines the role of SV as being a method by which students and the tutor can communicate their understanding of a program, as well as solitary working. The next section outlines the design requirements of ISVL.

DESIGN REQUIREMENTS

The design requirements underlying ISVL are taken from three sources: those governed by the pragmatics of the course, motivation from related empirical work, and practical issues relating to the underlying architecture.

Requirements From Pragmatics Of Course

Our students are studying part time and are required to fit their studies around other commitments, which means they will tend to study at different times. The Open University currently runs courses across wide areas of Europe and is starting to offer courses to non-European countries. Students therefore require systems which are:

(i) easy to use,

(ii) low cost,

(iii) multiplatform,

(iv) accessible from anywhere in the world, and

(v) enables asynchronous communication with their tutor and peers.

Principles From Experiments

In the theoretical section we described how experts were able to identify and strategically use implicit information from the SV display. Novices were unable to use such strategies, but did show the seeds of such a strategy when using TPM, in the form of what was termed an overview strategy. When using an overview strategy, the student would try and look for patterns in the display, though with little success. The richer collaborative environment will be used to support the student in developing the simple overview strategy into a form allowing more complex and abstract patterns to be appreciated. Such a strategy will help the student to view the program in more abstract and diverse ways. This will be particularly important when the student moves onto larger programs, with correspondingly larger execution spaces. Eisenstadt (1993) found that many difficult bugs are due to the distance between the bug cause and bug symptom. Locating such bugs requires the programmer to be able to appreciate larger patterns in the display. Scaffolding the student?s use of TPM with annotated sound which draws the students? attention to such patterns will encourage them to develop these skills, which will be crucial at higher levels of expertise.

ISVL also supports multiple representations of the execution to enable the student to take different perspectives depending on their task. Currently, ISVL features two SVs, TPM and Theseus, specifically chosen because of the different ways in which they present Prolog execution. Theseus shows a detailed textual account of execution near the code level. TPM provides more of a high level graphical account. Future studies aim to show how these two views can be used at various stages along the course, and how lessons learnt in one can be assimilated with the other view. ISVL also has a code highlighting option to emphasise the mapping between the visualization and the code.

Design Principles from the Web-server based Architecture

The fact the ISVL is composed of a central server and a client requires that we think about where the components should go. In general data which is on the client and is therefore local will be rapidly accessible. But local data increases start up time (because the Java code has to be downloaded). Storing structures on the server reduces the student requirements for processor power and memory and facilitates collaboration. This has to be balanced, however, with the fact that access to server structures are slower and have a ?dial-up?

cost.

The split we have used in ISVL is shown in figure 3.The program execution history and views are stored on the server. The history is made up of events which cause players to change state (see (Domingue et al., 1992) for details). A view consists of a series of locations (e.g. we have tree based and table based views). A mapping maps a history structure to an icon/image class. The views and maps form the basis of the ISVL HTTP protocol which is used to transmit the SV. The client interprets the view and map data to produce the on screen images which the user manipulates through a navigator. The navigator allows large execution spaces to be viewed by using techniques such as scale and compression.

Using a protocol specific to SV systems means that the amount of data transferred is minimal. Reproducing views and maps on the client means that an SV can be examined without the need for repeated accesses to the server.

INTERNET SOFTWARE VISUALIZATION LABORATORY (ISVL)

In this section we shall describe how ISVL embodies the approach outlined in the introduction, using a scenario involving a hypothetical student Bill Hoad and tutor Ingrid Johnson. Within the scenario we shall show how ISVL supports students debugging their programs and synchronous and asynchronous communication between students and their tutors.

Bill is working on an exercise from his course workbook. The exercise asks students to write a sorting program, called qsort, which uses the quick sort algorithm. The quick sort algorithm works by splitting a list around an element into a list of lower numbers and a list of higher numbers which are then recursively sorted. The program should take an unsorted list and return a sorted list. For example, the query:

where _y represents an unbound variable, should return:

signifying that the query has been proved and the value of _y is [1, 3, 4].

Bill loads his solution by copying it into the text area and clicking on the ?Load? button in the control panel (see figure 4 below). The following steps occur as numbered in figure 4:

1. Bill types his query:

```
qsort([4, 3, 1], _y)
```

and clicks on the ?Evaluate? button,

2. The result of the query ?NO? (indicating that the query has failed) and a TPM visualization are returned.

3. Bill steps through the execution using the button in the control panel until he reaches the point shown in figure 4 He notices the suspicious failure of the arrowed qsort node and investigates this further by clicking on the node to obtain a Theseus style fine-grained view (also arrowed). The finegrained view tells Bill that there is no match when the list to be sorted is empty. He fixes his code and tries the query again - this time it is successful.

Some time later Bill tries his code with the query:

```
qsort([4, 2, 3, 1], _y)
```

and gets the result:

```
YES

_y=[2, 1, 3, 4]
```

Text Area

Graphics Area

Control Panel

2. Result of Prolog Query

1. Prolog Query

3. Fine-grained view of

Figure 4. The ISVL code developement environment.

Even with the visualization Bill fails to track down the bug, so he calls up his tutor Ingrid. Ingrid agrees to start up a collaborative tutoring session. To start a tutoring session Ingrid needs to move to Bill?s work area and then start broadcasting. Tutors can connect to any of their student?s work areas by choosing their identifier from their current students page. Ingrid?s current student page is shown in figure 5 below. Using the mouse she chooses ?B. Hoad M251067? and her screen changes to one similar to figure 4. She then starts broadcasting by clicking on the ?Broadcast? button in the control panel.

Bill puts his ISVL into receive mode by clicking on the ?Receive? button. Ingrid?s interface actions are now replicated in Bill?s environment. Ingrid evaluates the query:

qsort([4, 1, 2, 3], _y)

turns on the fine-grained trace option and plays forward to the point shown in figure 6. She then says into her microphone:

?Things have gone wrong in the highlighted region (Ingrid highlights the region by dragging the mouse (see figure 7)). Splitting the list [3, 1] around 2 should yield the list [1] (numbers less than 2) and the list [3] (numbers greater than 2). I can see that this is the first time that greater-than (>) has succeeded, here (Ingrid rings the white ?>? node by clicking on the ?Trail? button and then drawing a circle using the mouse (see figure 8a)), and that the first split clause has been tried. The split here (Ingrid rings the

top split in red (dark in greyscale) (see figure 8b)) has put 1 in the wrong place - this comes from this split (Ingrid rings the lower split in yellow (light in greyscale) (see figure 8c)) so there?s probably a data flow problem in your second split clause. Let me check your code. Yes, if we compare your two split clauses (Ingrid brings up Bill?s source code) you can see that you?ve not been consistent in the order your two variables ?Highs? and ?Lows?.?

Bill can now use this help to continue with the exercise unaided. ISVL automatically records and stores broadcast sessions allowing Bill to retrieve and replay the commented visualization at any time.

Figure 5. The ISVL interface for moving to a student work area.

Figure 6. The ?crucial point? in the visualization which Ingrid plays to, so that she can explain Bill?s bug to him.

Figure 7. Ingrid highlights the problematic part of the execution on her machine and this is reflected on Bill?s interface.

(a) (b) (c)

Figure 8. Three annotations Ingrid uses in her explanation of the buggy execution to Bill. ringing (a) the first successful ?>? node, (b) the top split which returns an incorrect value (c) the lower split which has an incorrect call.

DISCUSSION AND CONCLUSIONS

The ISVL approach to integrating SV within the educational experience has a number of clear benefits:

? collaboration - either between students or with a tutor,

? synchronous and asynchronous communication - using live contact or archived material,

? platform independent - the client software is written in Java and will run on any Java aware browser,

? cost - no licensing costs for students, low distribution costs and no high specification machines required by students,

? detailed/quick evaluation - all student activity within ISVL can be stored and analysed centrally,

? reduce evaluation cycle - changes to ISVL in response to evaluation findings can be made quickly,

? enhanced tutor contact - a rich collaborative context supplements
email and telephone contact,

? intelligent central server - provides options for semi-automated
student support and data analysis.

ISVL will allow us to continue our research into the evaluation of
both SV and teaching over the internet. We have taught Prolog, Lisp,
and Knowledge Engineering over the internet and have experimented with
a virtual summer school for teaching psychology (Eisenstadt et al,
1996). Both the

prolog and knowledge engineering have used visualizations (TPM and TRI
(Domingue and Eisenstadt, 1989)) as a key component, though never
before delivered over the internet. This will extend our work to
incorporate an enhanced level of collaboration. Similarly, Brown and
Najork (1996) are working on an internet visualization called CAT
(Collaborative Active Textbooks) which shows programs on an
algorithmic level. Our own work is complementary to this, showing the
execution at the code level, with the advantage that the user can more
easily move from a tutoring to exploratory scenario, which is less
possible with animations on the algorithmic level which require a
working program.

With the ISVL environment we will be able to undertake a more
fine-grained evaluation, looking not only at usability issues but also
deeper issues of external representations and how they impact on
learning outcomes. The issues we intend our evaluation to address can
be posed as three separate questions:

1) What is the most effective use of SVs in education: cradle to grave
or stage dependent?

2) How will students collaborate and share resources when working in a
virtual teaching environment?

3) How will the tutor?s role differ when using ISVL compared to a
traditional setting?

We hope our forthcoming evaluation of the course using ISVL will
illuminate these areas.

ISVL is based on a decade of empirical work and software development and will be used and tested on a real course. Due to the localised nature of the materials on the ISVL server, empirical findings using ISVL will be able to feed directly into the future presentation of the course, as well as informing us as to how SV technology can be most appropriately incorporated into a computer programming curriculum.

REFERENCES

Baecker, R. M. and Sherman, D. (1981). Sorting Out Sorting. Narrated colour videotape, 30 minutes, presented at ACM SIGGRAPH ?81. Published by Morgan Kaufmann, Los Altos, CA.

Brayshaw, M. and Eisenstadt, M. (1991). A Practical Tracer for Prolog. International Journal of ManMachine Studies, 42, 597-631.

Brown, M. H. (1987). Algorithm Animation. Cambridge, MA: MIT Press.

Brown, M. H. (1991). Zeus: a system for algorithm animation and multi-view editing. In Proceedings of IEEE Workshop on Visual Languages. Kobe, Japan.

Marc H. Brown and Marc A. Najork. (1996). Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom. Proceedings of the IEEE Symposium on Visual Languages (VL'96), Boulder, CO, Sept 3-6, 1996.

Chi, M. T. H., Feltovich, P.J. and Glaser, R. (1981). Categorisation and representation of physics problems by experts and novices. Cognitive Science, 5, 121-152.

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P. and Glaser, R. (1989). Self explanations: how students study and use examples in learning to solve problems. Cognitive Science, 13, 145- 182.

Domingue, J. and Eisenstadt, M. (1989). A New Metaphor for the Graphical Explanation of Forward Chaining Rule Execution. In Proceedings of The Twelfth International Joint Conference on Artificial Intelligence (Detroit, MI), 129-134.

Domingue, J., Price, B. and Eisenstadt, M. (1992). Viz: A Framework for Describing and Implementing Software Visualization Systems. In Proceedings of NATO Advanced Research Workshop: User-centred requirements for Software Engineering Environments, September, 1991.

Eisenstadt, M., Breuker, J., and Evertsz, R. (1985). A cognitive account of ?natural? looping constructs. In B. Shackel (Ed.), Human-Computer Interaction. Amsterdam: Elsevier (NorthHolland), 1985.

Eisenstadt, M. (1993). Tales of Debugging from the Front Lines. Empirical Studies of Programmers V, Palo Alto, CA., December, 1993.

Eisenstadt, M. and Brayshaw, M. (1988). The Transparent Prolog Machine (TPM): an execution model and graphical debugger for logic programming. Journal of Logic Programming, 5 (4), 277-342.

Eisenstadt, M., Bayshaw, M., Hasemer, T. and Issroff, K. (1996). Teaching, learning and collaborating at a virtual summer school. In A. Dix and R. Beale (Eds.) Remote cooperation: CSCW Issues for Mobile and Tele-workers. London: Springer.

Gloor, P. A. (1992). AACE - Algorithm animation for computer science education. In Proceedings of Visual Languages Workshop, Seattle, WA: IEEE Computer.

Lieberman, H. and Fry, C. (1995). Bridging the gap between code and behaviour in programming. In Proceedings of CHI ?95 Human Factors in Computing Systems. New York: ACM Press.

Mulholland, P. (1995). A framework for describing and evaluating Software Visualization Systems: A case-study in Prolog. PhD Thesis, Knowledge Media Institute, Open University.

Petre, M. and Green, T. R. G. (1990). Where to draw the line with text: some claims by logic designers about graphics in notation. In Human-computer interaction INTERACT ?90. NorthHolland: Elsevier.

Stasko, J. T. (1990). Tango: A Framework and System for Algorithm Animation. IEEE Computer, 27-39.