A Visualisation Tool for the Programming Process

Charles Boisvert

School of Computing & Info. Systems,
Norwich City College
Ipswich road, Norwich NR2 2LJ, UK
(+44) 01603 77 3203

cboisver@ccn.ac.uk

ABSTRACT

eL-CID (e-Learning to Communicate Iterative Development) demonstrates computer programs' iterative design using computer animation. It translates descriptions of iterative editing into a dynamic visualisation of the changes, as if code was being edited in front of the user.

A range of animations has been developed and the system evaluated through action research. The evaluation shows that it is particulary useful as a reflective tool, revealing the problem solving inherent to development.

Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]: Computer science education.

General Terms

Algorithms, Design, Human Factors.

Keywords

Iterative programming, e-learning, visualisation

1. INTRODUCTION

Computer programs are not written in a vacuum: they are worked out through a process. Textbooks and written material, however, are limited to static examples, and so communicate this process poorly. Unlike a Mathematical calculation, where many simple steps can be written out to reveal the resolution process, the development of a computer program is too voluminous to be written in any but the scantiest detail and followed by a novice in traditional media.

The result is that, as Caspersen and Kölling [7] say, "texts appear to deal with program as a noun rather than as a verb". Yet the present media enable other approaches to communicating the programming process to novices.

Our system supports the learning of programming by visualising aspects of the development process. eL-CID (e-Learning to Communicate Iterative Development) presents a description of the programs' development as if code was being edited in front of the students.

An XML application is proposed to represent elementary program development. eL-CID translates this representation into an animation of the development for the learner. The system thus helps communicate software development by showing dynamically the program development process which is so difficult to present statically.

This paper first analyses the obstacles to showing the development process in print and reviews visualisation systems as alternatives. On this basis, it proposes a system based on an XML representation of text editing. It describes how the software supports the production of editing histories, its translation into animations, and the repository of examples that have been developed thus far. Finally, we evaluate the system with users, and show its benefits for teachers and learners.

ITiCSE'09, July 2009, Paris, France.

2. BACKGROUND: VISUALIZING SOFTWARE DEVELOPMENT

2.1 The curse of print

The difficulties of showing students the development process stem primarily from attempting to show statically --- in diagrams or in print --- a practice which is dynamic. As Bennedsen and Caspersen [2][3] write, "textbooks neglect the issue" of the programming process.

An indication of the limited success of showing the programming process in print is given by Kölling (reported by [3], op. cit.). In an invited lecture, tellingly entitled "the curse of Hello World", Kölling presents an analysis of 39 popular programming textbooks, showing that texts are structured according to language constructs rather than development techniques. This emphasis on example programs and code excerpts over problem solving extends to much of programming education; as Malan and Leitner [14] say, students have to be "masters of syntax before solvers of problems".

We show in [4] that teachers attempt to adapt the static medium of the page to communicate the process programming. New media may, however, support the communication of the programming process.

2.2 Software Visualisation Tools

A good survey of Software Visualisation (SV) was made by Gracanin et al. [11]. SV focuses on techniques to graphically represent data, program code, and algorithms.

Data visualisation is well established in research, industrial and educational systems. Recent work includes educational systems with proven capabilities such as BlueJ [13] and PlanAni [6].

Algorithm visualisation systems are used in teaching as they can show traces of complex execution (see for example, [16][15]). In

algorithm visualisation systems, the steps of execution are usually shown in an animation to allow a viewer to construct a mental model of how the process leads to the required result. This use of animation provided the initial insight for eL-CID: the same principle could be used to show, not the system's process, but elements of the programmer's process, revealing to a user how the program is developed.

Visualising program code is pervasive in software engineering. Particularly relevant to this paper is code (or software) evolution visualisation: the representation of how systems change over time. Data management is often used as in [1] to record the state of software projects and successive versions of source code to support teamwork, backup and versioning; visualisation tools also help overview large systems [9]. Seesoft, by Eick et al. [8], provides visualisations for systems of up to 50,000 lines of code. However, such systems focus on visualisation in the large, representing statistical data on bug density, revision frequency and the growth rates or number of lines of code of the system.

Although the transformation of a program in the iterative development process seems not to have attracted the interest of the SV community, SV systems have brought relevant advances in the representation of complex information. The tools and techniques used in algorithm visualisation could be adapted to visualising program development.

2.3 Visualising the development process

Starting with Papert's [17] development of the Logo language, many tutoring systems have been developed and used to facilitate learning of program development.

Bennedsen and Caspersen's [2] approach is to record the development process in video to offer the learners the description they need. They show that many notions can be captured by a process recording. Compared to development tools however, video recordings lack the flexibility of interactive systems.

A meta-study by Hundhausen et al. [12] showed that the success of pedagogical visualisation systems in experimental studies has more to do with the students mode of engagement than with the actual visualizations: active learning shows better results. This indicates that video process recordings, would be more beneficial still if learners had greater control over the visualisation.

One alternative which attempts to provide such flexibility is Viewlets. Viewlets, a commercial system [19], provide a record of the screen which can be annotated by the author and enhanced with some interactive features, such as interrupting the replay to let the user find a hot spot, or providing a written commentary. eL-CID proposes to offer similar flexibility. Our approach follows a similar line but attempts to make visualisation more flexible by recording elements that are particularly relevant to the programming process.

3. PROPOSED SYSTEM

An XML application describes the basic steps of code editing to model the development process. The system is composed of an editor for development examples, a viewer, and a web repository.

3.1 Describing and viewing development

The XML description of editing and the viewer are described in more detail in Boisvert [5].

To model elementary program development, we represent basic text editing actions (Cursor move, Insert, Select, Delete, Copy, and Paste) in XML, with optional annotations.

eL-CID is written in JavaScript using AJAX technology to load and parse the development history. When a development example is loaded, the system parses the initial source code and its incremental changes. The viewer presents the program as if it was being edited in front of the user. Students can also execute, edit or print the program and annotations at any stage, as shown figure 1.

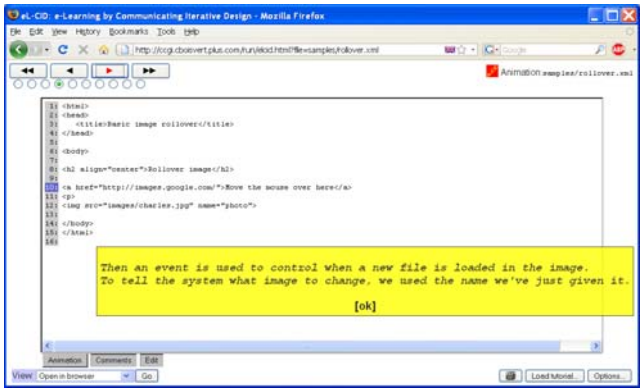

Figure 1: eL-CID viewer

This approach puts the user in control of the visualisation: they opt for what to view, to run, and to print, which could not be obtained from a video process recording.

3.2 Development tutorials editor

Figure 2 (below) presents the interface of our animation editing tool. Like the animation system, it runs entirely in a Web browser.

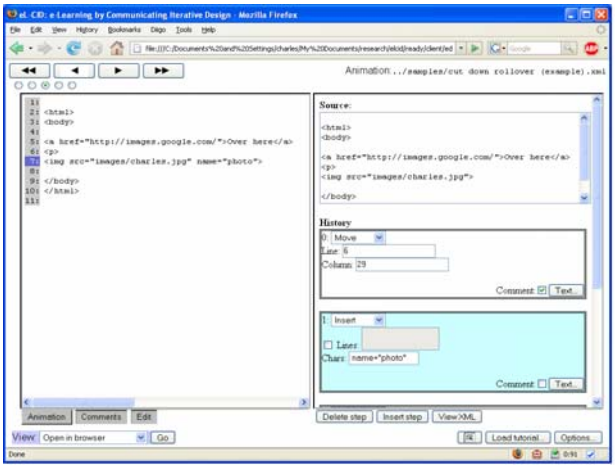

Figure 2: eL-CID editor

The visualisation is only useful with a volume of development tutorials; at present 30 examples are in use, covering a range of topics in web programming. This makes an editor indispensable.

The editing tool builds on the existing presentation system. A panel on the left-hand side shows the animation; on the right, the record of development is presented step by step, so that teachers are able to build, review and annotate animations visually.

The approach to the editor is straightforward. The characteristics of each modification are stored in a step object created using the object-orientation features of JavaScript. Fig. 3 (below) illustrates the functionality of the step object.

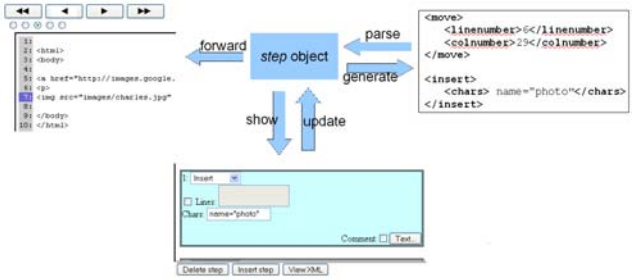

Figure 3: Methods of the step object

Each step holds the data needed to present a modification of the code and the optional comment; it also carries the methods necessary to translate it to and from different representations: visually into the animation; to and from an input form for editing; and to and from the XML file data for loading and saving. This lets the system co-ordinate both the display of the development process, and the editing interface.

When a tutorial is loaded, the system analyses the XML file to identify the source code, and create each step. The steps are then placed into a array to be shown in order.

With a larger number and greater diversity of tutorials available, there is a need to make them easy to find, organise and use. This is the purpose of a web site that could support a larger volume of tutorials and usage.

3.3 Website

The site is available at http://www.boisvert.me.uk/. Its is to act as a tutorial repository. Many repositories exist, including highly popular ones to share pictures, video images and other media as well as educational ones. These provided ideas for a web site that supports subject learning. Figure 4 (below) shows the site's main page, with many of its key features visible.

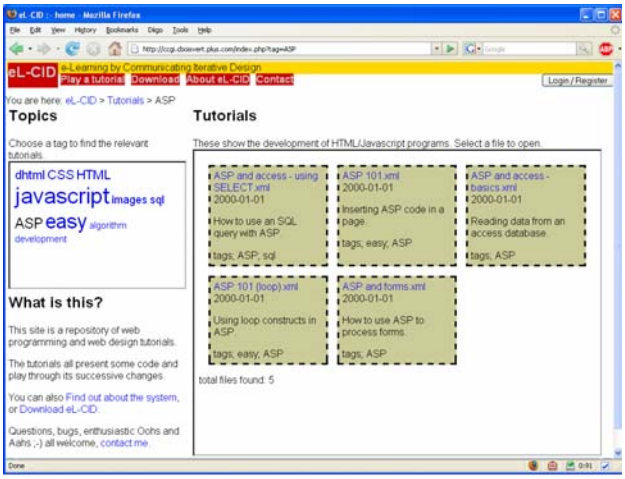

Figure 4: eL-CID home page

Each tutorial is presented in a clearly visible section, with a link and relevant information; this makes the set of tutorials easily scanned. Keywords are presented in a word cloud: larger links correspond to the number of visits, while the links are also ordered by how many tutorials are available for that tag. The two criteria help identify quickly more important keywords to search for. Finally, a breadcrumbs trail lets the user know where they are in the site's hierarchy, while acting as a navigation tool as each keyword is set as a link.

A database holds the information needed to create the multiple views, search through the tutorials available, and monitor tutorial usage.

The licensing terms of eL-CID were chosen to facilitate its development. The core visualisation system is available under the GNU/GPL license.

The XML data for the development examples, however, is excluded from the GPL. The examples developed thus far are in the public domain, but new examples developed by any individual or organisation can be exploited as they see fit – distributed freely or sold, licensed, or kept as exclusive training resources.

4. EVALUATION

The evaluation used an action research approach: students were taught using the new software. Concurrent research was used to study the system and its usage.

4.1 eL-CID in the classroom

A database tracked access to the examples and interface events to provide a rich source of information about the use of eL-CID.

The system was used with three groups of students in the spring semester of 2008. Data on student success is of very little value in this instance: students numbers are small, 37 across three groups not studying identical curricula; and as in all U.K. colleges, high student success rates are set as a target by funding bodies. As Prof. Strathern [18] has shown most clearly, a target ceases to be a good measure.

The number of viewings, their duration, the number of commands used, were collected and analysed for each tutorial available. Correlations are absent, as they were in an earlier trial [5].

We believe that the students adapt their usage to the range of resources available to them, and exploit them alternatively as references, to learn a technique, or to kick-start their own work with a suitable starting point, varying enormously their use.

This is as one might expect from a tool supporting constructivist, active approaches to learning.

4.2 Lecturers and eL-CID

The system was also shared with several lecturers to develop their own examples. The previous section highlights example development as a painstaking process. Making descriptions of the history of a development, however, also turns out to be a revealing exercise.

Some algorithms have a clear, well known development path which has been frequently used in teaching. One such example, which was developed as an example with eL-CID, is selection sort: there is a straightforward development from finding the largest element in a set, to finding the next largest, and to the

well-known concept of embedding loops to check every element against every other.

Such a clear development path is not necessarily the case, however, and writing development descriptions for animations in eL-CID is a revealing process, even on simple algorithms. An example will make this very clear.

The programming example (figure 5, in JavaScript) is commonly used to teach array manipulation. The program checks that the array of numbers is in ascending order. It is a good example of the difficulty students face when they are first learning to program.

```
var numbers = new
Array(673,623,63,54,34,23,9,4,1);

var sorted = true;

for (var i=1; i<numbers.length; i++)
  if (numbers[i-1]>numbers[i]) sorted=false;

if (sorted)
   alert("The array is in ascending order");
else
   alert("The array is not in ascending order");
```

Figure 5: programming example

Most students are able to analyse the example and understand how it works out the result; yet given the question "Write a program to check whether an array is in descending order", the same students that understand this program when given to them, have difficulties working it out for themselves.

Using eL-CID to view the resulting development, we wrote a description of one possible development path for this example. The path reveals a difficulty in the program that, for inexperienced students, is best worked out iteratively.

First a partial solution (figure 6) is proposed that checks that two elements are in order. That solution is of course incomplete but it is straightforward and it reveals some elements that are useful in the finished work.

```
var sorted = false;

if (numbers[0]<=numbers[1]) sorted=true;
```

Figure 6: working it out

To check the whole array, the program is modified by adding a loop (figure 7). The result is incorrect: it checks whether any pair of consecutive numbers is in descending order (rather than all pairs). Yet, working out that incorrect program is productive, as it reveals a difficulty in the exercise that is not evident to the expert. It also brings us close to a correct solution.

```
var sorted = false;

for (var i=1; i<numbers.length; i++)
  if (numbers[i-1]<=numbers[i]) sorted=true;
```

Figure 7: Incorrect, yet productive answer

Identifying the problem with (7) reveals a correct solution to the problem, by using the loop to identify where the array is not in order, rather than when it is.

```
var sorted = true;

for (var i=1; i<numbers.length; i++)
  if (numbers[i-1]>numbers[i]) sorted=false;
```

Figure 8: a correct program

Played in an animation, this development description shows a student how they could have worked out this solution. However, this example is of just as much value for a teacher. Working out the development path reveals to a teacher where the difficulties of this exercise are for novices: it is not self-evident that to find out whether an array is in order we should check where it is not.

Soon the better learners discover patterns or methods that they can apply to such problems and the initial difficulty turns to disbelief that this example can befuddle students. Yet eL-CID, by providing means to review how such a simple example is developed, reveals to teachers where the difficulties are in the development process.

5. CONCLUSIONS

The project started with a view to find better means to present iterative program development. The development and evaluation work to date shows that the value of the work is as valuable for the teachers, as a tool to reflect on practice, as for the students.

5.1 Iterative Development, revisited

The process of turning a problem into a working program encompasses many activities and skills. This system does not support learning all of them, but rather responds to one concern: when novice students are presented with anything but a trivial programming example, they understand solutions, but not, in the words of one student, 'how to get there'.

Many students lack the means to do what experienced programmers take for granted: solve the problem separately, then generate the code itself. So such students need practice to construct their problem-solving abilities, as Papert [17] identified that children do with Logo.

The example developed in the section 4 shows that useful development examples in that regard have three characteristics:

- Non-trivial; working out the area of a rectangle given its length and width does not involve iterative attempts!
- Partial answers are productive; a program that finds the largest element of an array is not a sorting algorithm - yet it is a productive step. Here we join the common technique of incremental development, which emphasises small steps of implementation and testing.
- Mistakes are productive. Students learn problem-solving by developing both model and implementation, experimentally. In that context, implementation mistakes commonly reveal problem-solving difficulties, and can be exploited by students to develop their problem-solving abilities.

5.2 Developing further examples

The criteria set out above form a class of examples that demonstrate the value of refining solutions iteratively, through small steps of implementation and testing, early in a curriculum, before students are able to follow more formal approaches to development.

More tutorials are needed that follow these criteria to support the development of problem-solving and programming abilities in beginning programmers. A few well-known examples are worth mentioning, that are simple enough to present to novice students, while satisfying the criteria set out above: non-trivial, productive

partial answers, productive mistakes. Here are some examples that satisfy these criteria:

- Developing the selection sort algorithm. There is a clear development path to this algorithm, from working out the largest element of an array, to the next largest, to using a loop to sort the full array.
- Creating a monthly calendar. A structure would display a correct table with days of the week, but mark the first of every month on a Monday; iterative development can lead to a correct implementation and reveal how to resolve the problem.
- Sliding Delta problems. David Ginat [10] describes a sliding delta pattern and gives several examples, showing that students who are unaware of it opt for less efficient solutions. Interestingly, the less efficient solutions make the pattern apparent: and so the mistake is productive.

EL-CID, by presenting the development of simple examples as animations, make it easier to communicate the problem-solving required in computing.

6. ACKNOWLEDGEMENTS

This work has been made possible thanks to support from Norwich City College, Anglia Ruskin University, SIGCSE, and the HE Academy. I would like to thank all those who have made this possible.

Throughout the research, the reception and encouragement of my colleagues has been a great source of enthusiasm. My thoughts go to my colleagues who have supported me and this work.

7. REFERENCES

[1] Barghouti, N., Emmerich, W., Schäfer, W. and Skarra, A. 1995. "Information Management in Process-Centered Engineering Environments", *Process-Centered Environments*, John Wiley and sons.

[2] Bennedsen, J., and Caspersen, M. 2005. Revealing the Programming Process. *Proceedings of the SIGCSE symposium on Computer Science Education*, St Louis (U.S.).

[3] Bennedsen, J., and Caspersen, M. 2006. Exposing the programming process. Bennedsen, J., Caspersen, M., Kölling, M. (eds.), *Reflections on the Teaching of Programming Methods and Implementations*. Springer.

[4] Boisvert, C. 2004. Supporting Program Development Comprehension by Visualising Iterative Design. *Proceedings of the IV'04 conference on Information Visualisation*, London, 2004.

[5] Boisvert, C. 2006. Web animation to communicate iterative development. *SIGCSE Bull.* 38, 3 (Sep. 2006), 173-177.

[6] Byckling , P. and Sajaniemi , J. 2006. Roles of variables and programming skills improvement. *37th SIGCSE technical symposium on computer science education* (Houston, Texas, USA). ACM, New York.

[7] Caspersen, M. and Kölling , M. 2006. A Novice's Process of Object Oriented Programming. In *Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA). ACM, New York.

[8] Eick, S.G., Steffen, J.L., Summer, Jr. E.E. 1992. Seesoft—a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering* 18(11):957–968

[9] Gall, H., Jazayeri, M., Riva, C. (1999). Visualizing software release histories: The use of color and third dimension. In *Proceedings of the IEEE International Conference on Software Maintenance*.

[10] Ginat, D. 2004. Algorithmic patterns and the case of the sliding delta. *SIGCSE Bull.* 36, 2 (Jun. 2004), 29-33.

[11] Gracanin, D., Matkovic, K., Eltoweissy, M. 2005. Software Visualisation. In *Innovations in Systems and Software Engineering*, Springer.

[12] Hundhausen, C., Douglas, S., Stasko, J. 2002. A Meta Study of Algorithm Visualisation Effectiveness. *Journal of Visual Languages and Computing*, 13(3): pp. 259-290.

[13] Kolling, M., Quig, B., Patterson, A., ROSENBERG, J. 2003. The BlueJ System and its Pedagogy, *Journal of Computer Science Education* 13 (4).

[14] Malan, D., AND Leitner, H. 2007. Scratch for Budding Computer Scientists. SIGCSE Bull. 39 (1), 223-227.

[15] Mukherjea, s.., and Stasko, J. 1993. Applying animation techniques for program tracing, debugging, and understanding. In *Proceedings of the 15th International conference on Software Engineering* (Baltimore, Maryland, United States).

[16] Mullholland, P. 1997. Teaching programming at a distance: the internet software visualization laboratory, *Journal of Interactive Media in Education*, Knowledge Media Institute, Open University. http://www-jime.open.ac.uk/

[17] Papert, S. 1980. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, New York.

[18] Strathern, M. 2000. The Tyranny of Transparency. British Educational Research Journal, Volume 26, Number 3, 1 June 2000 , pp. 309-321.

[19] Qarbon. Accessed 2009. *Viewlet Builder*. http://www.qarbon.com/

Columns on Last Page Should Be Made As Close As Possible to Equal Length