Web Animation to Communicate Iterative Development

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 reveals a great diversity in the patterns of usage of the animations among students. It also identifies directions for further development and work that eL-CID enables in program development cognition.

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

This paper describes the design, implementation and initial evaluation of eL-CID, a system to support learning by visualising iterative design. eL-CID (e-Learning to Communicate Iterative Development) takes examples of computer programs' iterative development and uses computer animation to show them to students. It furthers an approach to communicating programming and algorithm design to novices, that finds in the present knowledge media a useful expression of a awkward-to-transmit concept: that programs are not written in a vacuum, but that they are 'worked out', produced iteratively.

A program development history is composed using a specially devised language. eL-CID translates it into successive frames in an animated demonstration of the code's iterative development. The system thus helps expert programmers communicate their software development skills by showing a process that is difficult to present statically.

ITiCSE'06, June 26-28 2006, Bologna, Italy.

This paper first reviews the background of software tools for visualisation, development, and instruction in software development. It then analyses practices in teaching iterative development and proposes a development history description language. It describes how the development histories are written, how they are translated into animations, and the range of examples that have been developed thus far. Finally, we report on an evaluation of the system and conclude with a discussion of the future of this work.

2. Background: software visualisation, development, and instruction

Software tools are frequently used to support and help teach iterative program development. Two fields are especially worth reviewing for their relevance and for their use in tutoring: software visualisation systems and software development or engineering environments.

2.1 The Cognitive Science of development

Research in Human-Computer Interaction [14, 15] Computer-Assisted Instruction [16, 21] and Software visualisation [12, 5] all concords to show that communicable systems, systems that are built on cognitive models, are the key to successful knowledge communication.

Starting with Papert's [14] development of the Logo language, many tutoring systems have been developed and used to facilitate learning of program development. Parallel to this development, the close relation between development tools and cognitive models remains a major preoccupation of the research community [3, 18].

Studies of the psychology of programming inform researchers with cognitive models of program comprehension [9]; however, they show more interest in *program* comprehension than in program *development* comprehension. Bennedsen and Caspersen, in their work using video to teach the programming process [4], show that the scant attention to the development process extends to textbooks.

That very realisation signals the growth of interest in the development process, as does the range and newfound popularity of the models like Agile programming [1, 8] which purport to integrate the iterative, lightweight processes into development methodologies.

A system to mimic aspects of the software development process is, therefore, a timely pedagogical tool.

2.2 Software Visualisation Tools

Software visualisation (SV) focuses on the development of techniques to present data and algorithm execution. Many teaching, research and industrial systems use data visualisation, as the survey of object-oriented programming environments by Romero *et al.* [19] shows. Algorithm visualisation systems however, are more relevant to eL-CID for the techniques they employ that this project could exploit.

Good examples of algorithm visualisation are [5, 12, 13, 20]. Algorithm visualisation is used in teaching to show another awkward-to-transmit concept: program execution. 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. eL-CID also relies on an animation, to show not the execution process, but the development process.

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 Software Development Environments

Integrated development environments (IDEs) and software engineering environments (SEEs) support the development of complex systems. IDEs exist to support program coding in practically any language. SEEs extend their support to other aspects of the development process, such as UML modelling, process management [10], or co-operative development [7]. Many of these systems were developed for the industry, and in teaching they support larger scale projects and more advanced development rather than programming initiation.

Two aspects of IDEs/SEEs are nevertheless relevant to eL-CID and should be pointed out. First is the use of data management to record the state of software projects and successive versions of source code, which is currently used to support teamwork, backup and versioning [2]. Second is the development of refactoring editors. Originally created to facilitate iterative development and improve editing [11], they support semantic based program changes, such as renaming, creating, or moving variables and methods. The change of emphasis from syntactic to semantic-based edition support is significant for eL-CID.

Both of these aspects of IDEs and SEEs are worth noting for the future directions of eL-CID; this paper discusses them further in the final section.

3. Visualising Program Development

Bennedsen and Caspersen *op. cit.* [4] state that textbooks 'neglect the issue' of the programming process. The problem is that it is particularly difficult to represent in a static medium – such as a book – the dynamic process of software development. However, it is possible to map dynamic program changes to static representations.

Techniques to deliver programming concepts can be judged along two criteria:

- One is process vs. product orientation. This is the issue that textbooks find difficult to transmit, although some texts compensate for the static nature of print by relying on the development of a small number of examples throughout; we analyse this phenomenon in [6].
- The other is synchronicity: whether the delivery is live and relies on the presence of the students with their lecturer, or can be viewed by the learners in their own time, asynchronously.

Based on these two criteria we compare a range of delivery techniques. The techniques outlined by [4] and [6] are placed all together in a chart (Figure 1).

Among the more common delivery techniques – textbooks and

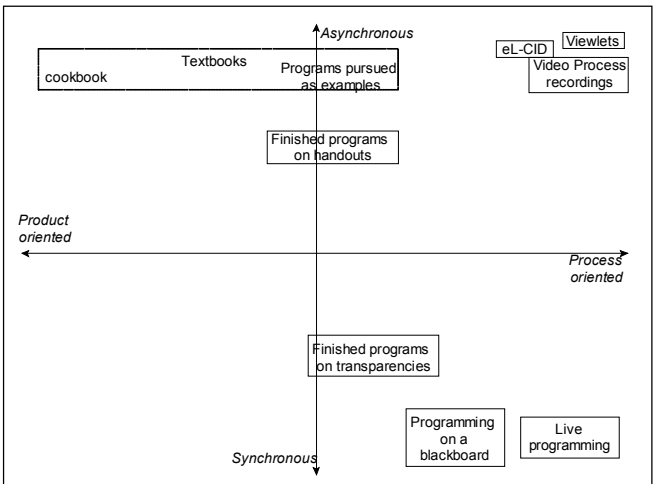

Fig. 1: comparison of delivery techniques

live lectures – a trade-off is apparent. Live delivery provides all sorts of means to reveal the programming process; if the students want to work independently, they face handouts, exercises and textbooks that are bound to emphasise the resulting programs.

Breaking that trade-off are process recordings: whether using videos, recording specific actions like eL-CID, or commercial systems like Viewlets [17]. These overcome the difficulties of showing dynamic changes in a static presentation, by using the computer to show the changes dynamically, ensuring a more natural mapping between the effective process of iterative design and the view that is proposed to the students.

4. Research and Implementation

To model the iterative development process, a descriptive language and XML application are proposed that describe the basic steps of code editing. This application is then used to model the iterative development of some well-known programming examples. A computer system was built that uses those descriptions to show iterative software development.

4.1 A simple language for iterative editing

Code editing can straightforwardly be reduced to syntactic changes. For eL-CID, six elementary changes are proposed:

- Cursor move: Move the cursor to a given position;
- Insert: Insert given lines and characters at the current position;
- Select: Select the given number of characters and lines;
- Delete: Delete the current selection;
- Copy: Copy the current selection to a clipboard;

- Paste: Insert the clipboard contents at the current cursor position.

These six operations suffice to model any changes to a text file. Stored alongside the source code in an XML file, they describe program modifications.

These six editing operations are not the only possible choices. The shortest list excludes the select, copy and paste operations, but these are now so common that they have been included to model the editing process.

Other operations could be added that reflect key presses, such as back and forward delete and up, down, left, and right cursor moves. The possible benefits of these operations did not justify their integration in an initial version of eL-CID.

4.2 EL-CID implementation

An expert programmer writes his or her development history in an XML file and optionally annotates the process. When eL-CID opens the file, it displays the initial program listing and shows different options available to the user: to move forward or back one step at a time, play the development as a timed animation, or instantly show the initial or the final program version. The screenshot (figure 2) shows the interface of eL-CID.

Each step is played forward at the student's request so that the

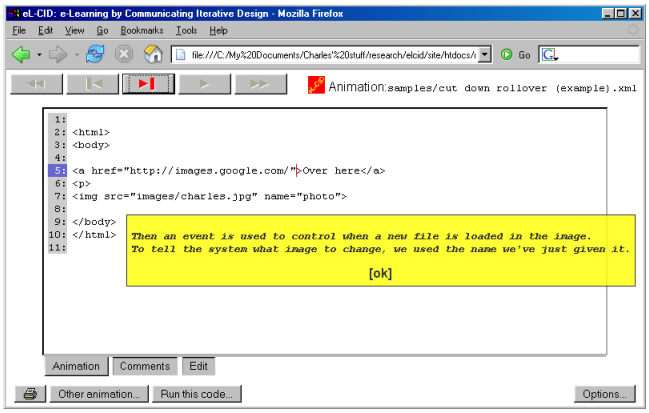

Fig. 2: eL-CID interface

changes appear on screen as if the expert was editing the code in front of the student. The optional annotations are displayed in a text box (a 'post-it') that can be moved away, turned off or aggregated with other comments for printing. At any step the user can stop to observe the changes and run the work in progress. They can also switch to an editing mode and continue the development process for themselves.

eL-CID is written in JavaScript using the AJAX technology to load and parse the XML development history. When a development example is loaded, the system parses the XML data into initial source code and its incremental changes. The successive changes are stored in an array of step objects, each step holding the characteristics of the change, the optional comment, and a forward method which is called when the step is shown on screen.

This approach allows a diversity of visualisations, putting the user in control of the visualisation as they opt for what to see, to run, and to print. This could not be obtained from a video process recording. However as [4] point out, video recordings allow the showing of a wider range of development process elements; the cost is poorer user control, offering an interesting complementarity of the two approaches.

4.3 Design examples, availability and licensing

eL-CID can demonstrate examples of iterative program development in any programming language. Four series of examples have been developed for web programming:

- HTML Web page development;
- Basic javascript programming;
- Web page animation examples;
- Array manipulation, selection sort;
- Basic Active Server Pages

In all, 15 examples are available to the students. The focus on web site programming is to build a resource on a consistent topic; the visualisation could apply to many other languages.

The software and examples developed so far are freely available from a website: http://www.boisvert.me.uk/

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.

5. Evaluation

The primary purpose of evaluating eL-CID is to determine whether it is useful in teaching. It should also help set priorities for further development and identify appropriate teaching practice.

Due to the nature of the tool and the environment 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.

5.1 eL-CID in the classroom

eL-CID was initially tried on one further education (pre-University) group studying Web Site Development, including an introduction to JavaScript programming, in the second semester of 2004. Examples were devised to match their curriculum and in response to the students' needs as these became apparent.

A multiple choice test to compare the eL-CID group against another learning with traditional methods proved invalid (students cheated to obtain top scores!). However even before the evaluation had run its course, it was clear that a finer analysis was needed of the students changing skills and the role eL-CID may play in that. Are their skills transferable? How well would they scale to larger, more complex problem-solving? Web design exerts a particular fascination on those students; would the same results appear in more arduous, computer science topics?

Feedback from the students showed that of eL-CID was well received, with students using the examples in their own work and developing their own scripts on the basis of the examples

from the web pages. In the words of one student, eL-CID shows "how to get there".

5.2 Widening use and refining the evaluation

As the stability and popularity of the system became clear, it was introduced to a growing number of students. The sucess of eL-CID with students caught up with the limits of our ability to develop program examples, which require patience and skill. Even for programs that have a clear development path producing new examples is a time-consuming process.

With the extending use a database was established to record usage information from the web site. We tracked access to the examples but also interface events to provide a rich source of information about the use of eL-CID in the hope determine, not just a one dimensional usefulness measure, but also an outline of what a tool like eL-CID is good for and in what conditions.

Twenty Higher Education students in two groups used eL-CID in the November-December period when data collection took place. EL-CID was one of the resources proposed to them, along with other tutorials and reference web sites, and with print resources. In the five weeks 11 November to 15 December there were thirty-three separate sessions on the site. Half of them were from the college and the other half from home.

The remarkable fact that emerges from the data is the diversity of the usage. Many sessions were short (13 under one minute), but the longest four stand out at over twenty minutes. All the commands available were explored by the visitors, although not on the same examples, some leading visitors to forward through the animation while others were loaded and the first or last frame

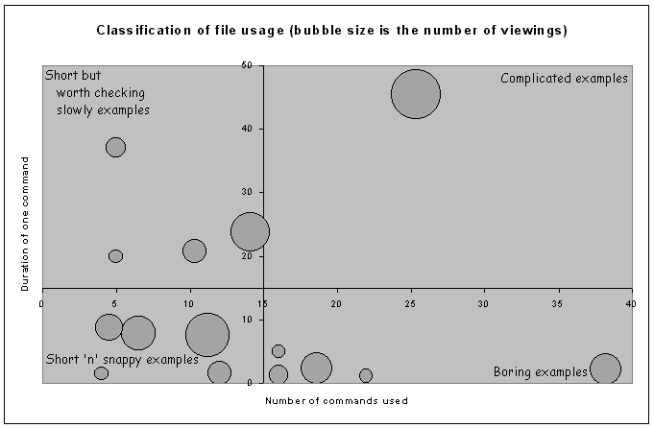

fig. 3: File usage on eL-CID Web site

brought for editing straightway. A graph of the usage by file (figure 3) will evidence this diversity.

The graph classifies the development examples according to two criteria: how many commands were used to view them, and how long (in seconds) each command took on average. If for instance an animation is low on both counts, it must mean that the user(s) loaded it, pressed few commands quickly, and moved on. The size of the blob is the number of separate viewings which gives an indication of the example's popularity. The graph divides the examples into four categories:

- *Short but worth checking* examples are those that require few commands but that users take their time over. These include javascript animation examples and an introduction to HTML.
- *Complicated* examples are the next quadrant, of files which are viewed using a lot of commands, slowly. There is only one such example, selection sort.
- *Short'n'snappy* examples, being viewed quickly with few commands, would be expected to be the less interesting, but – bubble size indicate return visits. They are image examples, an example of bringing a web page to world-wide web consortium standards, and a prequel to selection sort, finding the largest of an array.
- Finally*, boring* examples may not be so – they are files that were viewed using a lot of quick commands. There is an HTML table example, examples of ASP development, and one on forms and their JavaScript handling and on opening separate windows in JavaScript.

The correlations between number of viewings, their duration, the number of commands used, and the topics are conspicuous by their absence. In the early trials of eL-CID it appeared that the best examples were those with the clearest identifiable path through the program development. With a slightly improved interface, there is no such selection: the students seem to adapt their usage to the range of resources available to them, and exploit them as references, to learn a technique or to kick-start their own work with a suitable starting point, varying enormously their use.

6. Conclusions

The project started with a view to improve the demonstration of iterative program development. The development and evaluation work to date confirms the value of the approach.

6.1 Recording Development Histories

To be used more widely or adapted to the teaching needs of a wide range of different computing courses, more program development examples are required. The priority is to facilitate the building of program development histories quickly and accurately. An editor could be adapted or developed for that purpose; alternatively, development history recorded in development environments could be transformed in eL-CID's format for visualising. Ideally, eL-CID could be integrated to an editor, allowing for the integrated view of another's development work while working at the development interface.

Further, the eL-CID website should facilitate the exchange of development examples. File management and feedback and discussion tools would help leverage the value of the visualisation system.

The use of development history recordings as data would open new perspectives: students themselves may be able to record, review and share their path to program development, adding with their tutors to a body of development history examples. It is remarkable that in many other complex tasks, like for instance Mathematics, showing how solutions are worked out is seen as a natural, required part of the process, yet in computing this is rare.

Yet this purpose has a prerequisite: to progress from syntactic code changes to semantic changes.

6.2 Semantic chunks in program development

Many recurring patterns in code editing identifiable from the eL-CID examples point to semantic changes. Some of these can improve the eL-CID visualisation, such as these editing operations that are obtained by compounding the basic ones:

- Cut = copy + delete
- Move text = move + select + cut + move + paste

It may prove useful to eventually include shorthand notations for such common operations; an analysis of which basic combinations recur frequently would prove useful.

Refactoring actions, such as renaming, creating, or moving variables and methods, could be used to identify semantic elements of code changes. This would allow the development of more flexible visualisations that focus on the syntax or the semantics of the program development at the user's request. This also opens the possibility of using eL-CID recordings and usage data as a means to study empirically the novice developers' approaches to programming.

eL-CID is a helpful visualisation system; the evaluation described in this paper shows the need for better examples and some means to make them. In future, with recordings of students' program development, it may also offer a means to investigate the psychology of program development comprehension.

7. Acknowledgements

This work has been made possible thanks to support from Norwich City College, Anglia Ruskin University, and SIGCSE. I would like to thank all those who have made this possible: Rob Fiddy, David Lovell-Badge and Liz Magem at City College; Genny Gilbert at ARU; and Sally Fincher at SIGCSE.

Throughout the research, the reception and encouragement of my colleagues has been a great source of enthusiasm. My thoughts go to all the staff at the School of Information Systems as well as Tom Sott and Bob Cole at ARU who have supported me and this work.

8. REFERENCES

[1] Manifesto for Agile Software Development. http://www.agilemanifesto.org/

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

[3] Ben-Ari, M. (1999). "Bricolage Forever!", Workshop of the special interest group on the psychology of programming, University of Leeds.

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

[5] Boisvert, C. (1995). "A Learning environment for Natural Language Processing", proceedings of the CS-NLP'95 conference on the Cognitive Science of Natural Language Processing, Dublin, 1995.

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

[7] Fougères, A., Canalda, P. and Chatonnay, P. (2002). "Pédagogie de Projets tutorés Basée sur la Synchronisation de fragments de Procédés Coopératifs: Motivation, Modélisation et Expérimentation", editor, *Conférence ARIADNE*, Eddy Forte ed., INSA Lyon, France

[8] Fowler, M. (2003). "The new methodology", http://www.martinfowler.com/ articles/newMethodology.html

[9] Green, T. (2000). "Instructions and descriptions: some cognitive aspects of programming and similar activities", Invited paper, in Di Gesu, V., Levialdi, S. and Tarantino, L. (Eds.) *Proceedings of Working Conference on Advanced Visual Interfaces* (AVI 2000). New York: ACM Press, pp.21-28.

[10] Gruhn (2002). "Process-centered software engineering environments – a brief history and future challenges", *proceedings of the annals of Software Engineering*, 14, pp. 363-382, 2002, Kluwer.

[11] IntelliJ (2002). "IntellijIDEA overview", Jetbrains http://www.intellij.com/

[12] Mullholland, P. (1997). "Teaching programming at a distance: the internet software visualization laboratory", *Journal of interactive media in education*, KMI. http://www-jime.open.ac.uk/

[13] Mullholland, P. (1998). "Sharing Programming Knowledge over the Web: the Internet Software Visualization Laboratory", Marc Eisenstadt and Tom Vincent (eds.), *the Knowledge Web: Learning and Collaborating on the Net*. Kogan Page.

[14] Norman, D. (1986). "Cognitive engineering", In D.A. Norman and S.W. Draper (eds.), *User Centred System Design*. Hillsdale, New Jersey: Lawrence Erlbaum.

[15] Norman, D. (1988). *The design of everyday things*. ed. MIT Press 1988.

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

[17] Qarbon (2005). *Viewlet Builder*. http://www.qarbon.com/

[18] Retowsky, F. (1998). "Software reuse from an external memory: the cognitive issues of support tools", *10th annual workshop of the special interest group on the psychology of programming*, Knowledge Media Institute, Open University.

[19] Romero, P., Cox R., du Boulay, B. and Lutz, R. (2002). "A survey of external representations employed in Object-Oriented programming environments", *Journal of Visual Languages and Computing (Special Issue on Program Visualization)*.

[20] di Scala, R. (2000). "Un package interactif d'assistance par ordinateur appliqué à un enseignement d'initiation à l'informatique-programmation", *Presentation a l'Atelier conception de contenus pédagogiques du colloque international TICE 2000*, 18-20 octobre 2000.

[21] Wenger, E. (1986) *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann Publishers.