A tutorial to teach tracing to first-year programming students

Article · January 2012

CITATIONS
2

READS
583

3 authors, including:

Marthie Schoeman
University of South Africa
9 PUBLICATIONS **34** CITATIONS
SEE PROFILE

Helene Gelderblom
University of Pretoria
40 PUBLICATIONS **244** CITATIONS
SEE PROFILE

A tutorial to teach tracing to first-year programming students

M. Schoeman*
e-mail: schoema@unisa.ac.za

H. Gelderblom*
e-mail: geldejh@unisa.ac.za

E. Smith
*School of Computing
University of South Africa
Pretoria, South Africa
e-mail: esmith@artiflex.co.za

Abstract

Introductory programming courses typically have a bimodal distribution, with a low pass rate, a high drop-out rate as well as a high number of distinctions. This article reports on an attempt to address the low pass rate situation in a first-year programming module (COS1511) at Unisa by providing the students with an interactive tutorial to teach them how to trace a computer program. This is part of a larger research effort to create a framework of guidelines for teaching programming in an ODL environment by using visualization. Design science research methodology is used, and this article specifically addresses the research question: *What guidelines for the design and implementation of visualizations can be extracted from the development and initial testing of the tutorial?* We describe the development process and the structure of the tutorial and justify design decisions. We also discuss lessons learnt during the initial testing of the tutorial and its development, followed by guidelines, based on this experience. We also describe future research on the tutorial.

INTRODUCTION

The results of introductory programming courses typically have a bimodal distribution, i.e. a low pass rate, high drop-out rate as well as a high number of distinctions (Bornat, Dehnadi and Simon 2008; Corney 2009; Robins 2010). Unisa is no exception, and it being an open distance learning (ODL) institution, may even exacerbate this effect.

There are various approaches to address this problem. Researchers attempt to determine the reasons for this phenomenon as well as remedy it. This leads,

 ISSN 0256-8853 Progressio 34 (3) 2012 pp 59–80

among other things, to research on programming ability/aptitude, teaching and learning theories and models, teaching/learning and assessment tools and techniques, as well as the curriculum and typical problems first-year programming students experience (Milne and Rowe 2002; Lahtinen, Ala-Mutka and Järvinen 2005; Dale 2006; Kinnunen, McCartney, Murphy and Thomas 2007; Simon, Carbone, Raadt, Lister, Hamilton and Sheard 2008; Sheard, Simon, Hamilton and Lonnberg 2009).

This article reports on an attempt to address the low pass rate scenario in a first-year programming module (COS1511) at Unisa, by providing the students with an interactive tutorial to teach them how to trace a computer program. We describe the development process and the structure of the tutorial, and justify design decisions. We also discuss lessons learnt during the initial testing of the tutorial and its development, followed by guidelines based on this experience. The long-term goal of the research effort is to create a framework of guidelines for teaching programming in an ODL environment by using visualisation. Therefore, this article presents a study of a development process, not a study of a product.

BACKGROUND AND LITERATURE REVIEW

The bracelet project

One research initiative that considers the teaching of programming is the well-known BRACElet project (Lister etal. 2004; Lister etal 2009; Whalley and Lister 2009; Clear, Philpott, Robbins and Simon 2009/2010 Dec/Jan; Clear etal 2011). The BRACElet project is a multi-institutional, multi-national study investigating novice programmers' acquisition of programming skills. This project originated in New Zealand in 2004 with the aim of exploring students' skill in basic syntax, tracing code, understanding code and writing code, and the relationships between these skills. More than 30 papers by more than 20 (co) authors have been published.

Studies within the BRACElet project suggest the possibility that a novice programmer develops skills in a hierarchical order, as follows:

1. The novice first acquires the ability to read, i.e. trace, code. Once this competency becomes reliable,

2. the capacity to explain code develops. When students are reasonably capable of both reading and explaining,

3. the skill to systematically write code emerges. The skills in this hierarchy do not necessarily develop in a strict order, but instead, reinforce each other and develop in parallel. Furthermore, students require a certain minimal competence at tracing and explaining before exhibiting a minimal competence to systematically write code (Lister et al. 2009; Lister, Fidge and Teague 2009; Venables, Tan and Lister 2009; Whalley and Lister 2009).

Tracing

Several other studies confirm that students who cannot trace code, usually cannot explain code, and also that students who perform reasonably well at code writing tasks can typically both trace and explain code (Lopez, Whalley, Robbins and Lister 2008; Lister et al. 2009). In an ODL environment such as Unisa, where students have to study independently, the ability to trace, in order to debug and to understand what a program does, is all the more important.

Cogan and Gurwitz (2009) claim that teaching students tracing skills early in introductory programming courses improve the through-put rate for their courses. They specifically refer to memory tracing, as opposed to hand-checking a program. During memory tracing a programmer 'executes' a program(segment) on paper by tracing the statements one by one, and in the process, tracks the values of all variables and how they change during program execution. The purpose of this action is typically to debug a program, or to understand what the program does. With hand-checking the calculations to solve a problem is done independently of the program, and these results are then compared to the program's final output.

Thomas, Ratcliffe and Thomasson (2004) also concluded, after an experiment on their first year programming students, that students have to construct diagrams to trace code themselves in order to make sense of the code. Furthermore, if students discover the value of constructing tracing diagrams themselves, it encourages them to continue doing so.

Mselle (2010) found that a variation of trace tables, Random Access Memory (RAM) diagrams, improved program comprehension and programming skills.

Visualization to enhance comprehension

One of the tools used to enhance the comprehension of programming concepts, is software visualization or animation. This typically takes the form of either algorithm visualization or program visualization. Algorithm visualization, e.g. a sorting algorithm, will demonstrate what a program or algorithm does and how it is done, while program visualization, e.g. a debugger to assist with tracing a program, will show the contents of the internal memory during program execution.

Feedback on the success/effectiveness of visualization is mixed. Despite the general perception, research shows that visualization is not always pedagogically effective or used effectively (Shaffer, Cooper and Edwards 2007; Foutsitzis and Demetriadis 2008). Although a considerable number of algorithm visualizations have been developed, few of them are used by teachers other than the developers themselves. Various reasons are cited, such as the difficulty and lack of time to adapt the visualization according to the study material used at different institutions Naps etal. 2003; Levy and Ben-Ari 2007). The bulk of visualisations are algorithm visualisations.

A number of program visualization tools intended to teach foundational programming courses have been developed and studied extensively. Well-known open-source examples include VIP (Isohanni and Knobelsdorf 2010; Isohanni and Knobelsdorf 2011; Virtanen, Lahtinen and Järvinen 2005), Jeliot (Ben-Ari etal 2011; Čisar, Radosav, Pinter and Čisar 2011) and ViLLE (Kaila, Rajala, Laakso and Salakoski 2009; Kaila, Rajala, Laakso and Salakoski 2010; Rajala, Laakso, Kaila and Salakoski 2007; Rajala, Laakso, Kaila and Salakoski 2008).

Distance teaching of programming

Distance teaching of programming typically involves using a learning management system (e.g. WebCT, Moodle, Blackboard or SAKAI), as well as tools such as e-mail, forums, podcasts, vodcasts, chat rooms, wikis and blogs, see e.g. (Reeves, Baxter and Jordan, 2002). Sometimes automated assessment tools are also used. Frequently, the same model is followed for both onground and online courses (Meisalo, Sutinen and Torvinen 2003; MacDonald, Dorn and MacDonald 2004; Bruce-Lockhart and Norvell 2007; Malan 2009). This includes providing both printed and web-based materials, regular assignments with discussions on solutions, and tutors (Meiselwitz 2002; Meisalo, Sutinen and Torvinen 2003).

Additional approaches investigated for teaching programming online include, among others, (1) an electronic Adaptive Book composed of learning objects (LOs), which can be customised by instructors to deliver just-in-time interactive learning modules to students (Adamchik and Gunawardena 2003; Adamchik and Gunawardena 2005), (2) a virtual workplace that provides an online IDE and programming environment for students, which also allows the instructor direct access to the student's workplace (Ng etal. 2005), (3) online environments to provide synchronised interaction between students and instructors (Cavus, Uzunboylu and Ibrahim 2007; Domingue and Mulholland 1997; El-Sheikh, Coffey and White 2008), (4)blended learning (Robbins, 2008; Tsai, Shen and Tsai 2011) as well as collaborative learning efforts (Tsai 2012).

Using visualization to enhance comprehension in distance teaching can take several forms. Some institutions use videos to provide the classroom experience (Bruce-Lockhart and Norvell 2007; Murphy, Phung and Kaiser 2008). Others recreate the classroom experience with virtual lectures and Web-conferencing (Koifman, Shimshoni and Tal 2006; Bower 2009; Malan 2009). As in face to face teaching, both algorithm and program visualizations are used at times. Koifman, Shimshoni and Tal (2008) incorporate an algorithm visualization system with virtual lectures and web conferencing, while Bruce-Lockhart and Norvell (2007) use program visualization with interactive course notes. Jeliot3 has also been used in an online course (Čisar et al. 2011).

However, the high cost of bandwidth in South Africa, the large student numbers (approximately 3000 per semester), the time constraints of the semester system, inadequate Internet access and the technical proficiency level of first-year students (Ragan 2003 in Ury 2005) make many of the additional approaches mentioned earlier difficult to implement successfully in the Unisa environment. Including existing programming visualization tools in the course material would require students to learn another environment in addition to the prescribed IDE and compiler. Therefore, in our approach to enhance the online teaching of foundational programming at Unisa, we developed a customized program visualization tutorial to teach students how to draw variable diagrams. This tutorial is based on the Study Guide for COS1511, since correspondence between the artefact and the related study material is a key factor in its acceptance as a learning tool (Naps et al. 2003; Röβling 2010). Although the tutorial can be used online on myUnisa, it is also distributed on CD as part of the study material for COS1511.

Research approach

Purpose of the research

This research is an attempt to improve the pass rate of first-year programming students in COS1511 at Unisa by providing them with an interactive tutorial to teach them tracing skills. To give an indication of the extent of the problem, pass rates for COS1511 from 2008 to 2010 are shown in Table 1. As discussed in the sections on the BRACElet project and Tracing, tracing has been identified as an important skill in the ability to program. In the long-term the research aims to determine whether the tutorial enhances comprehension in such a manner that it assists students in mastering the art of programming to the extent that it can be measured in the throughput for the module.

Table 1: Pass rates for COS1511 from 2007 to 2010. Active students are students with examination admission which is based on assignments submitted. COS1511 changed from a year module to a semester module in 2009 (which explains the drop in pass the pass rate)

Year	Enrolment	Active students	#students wrote	# passed	% Passed/ Active	% Passed/ written
2008	3478	3270	2527	583	18	23
2009(1)	2096	2096	1472	380	18	26
2009(2)	1984	1984	1306	204	10	16
2010(1)	2145	1988	1362	323	16	24
2010(2)	1826	1644	1182	278	17	24

The key features of and the pedagogical benefits of visualizations are not clear (Urquiza-Fuentes and Velázquez-Iturbid, 2009). By studying the development process and evaluation of the tutorial itself and how students use it, we intend to derive guidelines for using visualization to teach programming in an ODL environment. The long-term goal of our research effort is to create a framework, enhanced by Human Computer Interaction (HCI) principles, for knowledge transfer with visualization to teach programming in an ODL environment.

Research question

This research article describes the development of the tutorial to teach tracing, as well as the product itself; and presents lessons learnt during this process.

Though the tutorial is part of a bigger research process, the research question addressed in this article is:

> *What guidelines for the design and implementation of visualizations can be extracted from the development and initial testing of the tutorial?*

Methodology

The bigger research effort follows the Design Science Research Methodology (DSRM). Hevner and Chatterjee (2010, 5) describe design science research as 'a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem.' The design science process follows six steps: (1) problem identification and motivation, (2) definition of objectives for a solution, (3) design and development, (4) demonstration, (5)

evaluation, and (6) communication (Hevner and Chatterjee 2010). Vaishnavi and Keuchler (2004) in Oates (2006) describe the iterative process involved in the design and creation of the research as follows: (1) awareness of a problem, (2) suggestion for a solution, (3) development of an artefact to address the problem, (4) evaluation of the artefact, (5) and conclusion where the results from the design process and the knowledge gained are consolidated and communicated.

The problem identified in the overall study is the bimodal distribution of the COS1511 pass rate, with the corresponding motivation to improve the situation. Objectives for the proposed solution are to improve the students' skill at tracing computer programs, with the aim of developing their programming ability and ultimately increasing the pass rate for COS1511.

The artefact intended to address the problem, is the tutorial described in this article. The typical purpose of program visualizations is to show the contents of the internal computer memory during program execution. In contrast, the objective of this tutorial is to teach students how to draw variable diagrams so that they can trace programs manually. The design of the tutorial as described in the section on Design decisions, is aimed at assisting the students to develop a mental model of how to keep track of what happens in computer memory by drawing variable diagrams.

To demonstrate the use of the tutorial, it is incorporated in the students' study material with compulsory assignments. Evaluation of the tutorial will take several forms, as described in the section on future research. This will provide the opportunity to consolidate and communicate the knowledge gained from the design and development of the tutorial.

Purpose of the tutorial

At Unisa, memory tracing and parameter transfer are taught in the introductory programming course, COS1511, by means of static variable diagrams in the study guide. We noticed that students do not understand the static diagrams in the study guide, but that face to face explanation of the static variable diagrams help.

In marking examination scripts for the follow-up first year module, it was also clear from the kind of errors students make that they lack knowledge and comprehension of the processes that occur in CPU memory during program execution. This knowledge gap is even more pronounced with regard to parameter transfer. This is supported by literature (Milne and Rowe 2002; Bruce-Lockhart and Norvell 2007), and confirms the need for a better understanding of what happens in memory.

This tutorial is intended as an aid to teach students how to do memory tracing, an important skill in the art of programming (Thomas, Ratcliffe and Thomasson

2004; Lopez et al. 2008; Cogan and Gurwitz 2009; Lister et al. 2009; Lister, Fidge and Teague 2009; Venables, Tan and Lister 2009; Whalley and Lister 2009). The tutorial shows how to draw variable diagrams to keep track of what happens in computer memory. It also demonstrates parameter transfer.

Since memory tracing is not machine or language dependent, it offers some of the same benefits as RAM diagrams proposed by Mselle (2010) such as portability, scalability and flexibility. Similarly, it can also be used to study and design code, as well as for debugging.

Tutorial specifications

The tutorial is developed in Adobe Flash Player 10 ActiveX. The screen dimensions fit a standard screen of 1024 by 768 pixels. The tutorial consists of 21 activities. The activities range in length from 7 to 60 steps. In total the 21 activities contain 600 steps.

The introductory frame provides a menu that allows users to choose an activity, to view the instruction manual or to exit from the tutorial. A small video demonstrating how to use the tutorial acts as instruction manual.

Activities and implementation of activities

Each activity demonstrates the tracing of a program or program segment by drawing variable diagrams, statement by statement. Each program or program segment presents a specific programming concept, e.g. reserving memory space for variables, or a programming construct such as a loop.

Screen and boxes

As the program is traced, each statement is explained. To explain a statement, the screen is divided into four panels, as shown in the diagrammatic representation of the layout in Figure 1: one each to show the program code, the computer memory, the program output and an explanation of the statement. The purpose of each panel is briefly explained inside the corresponding box in the diagrammatic representation.

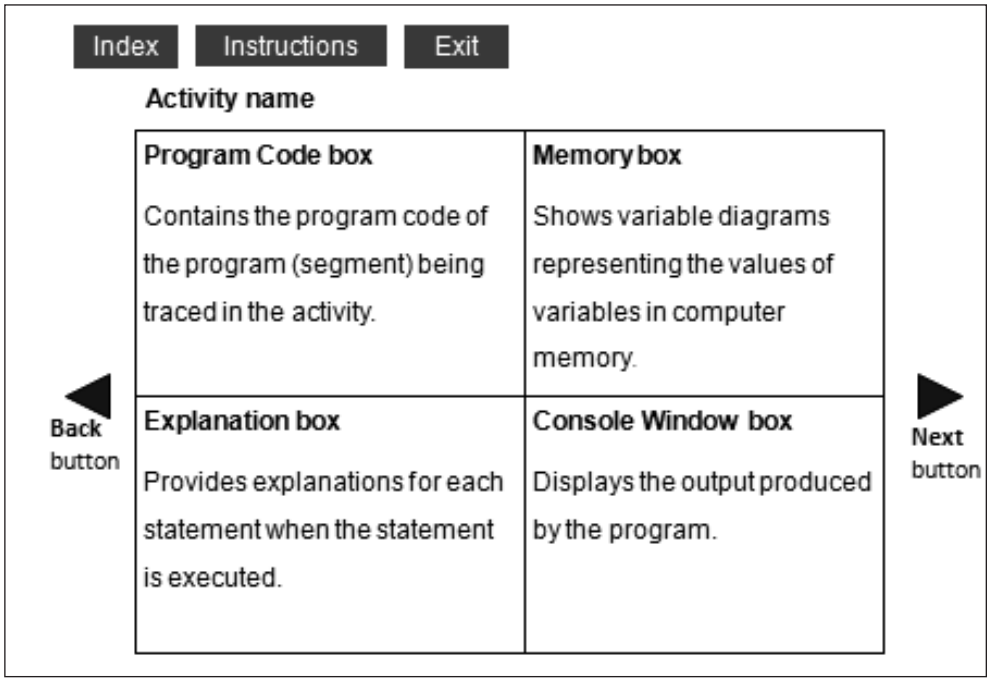

Figure 1: Diagrammatic representation of screen layout for tutorial, presenting the purpose of each panel. The menu appears on top, and the navigation buttons (Back and Next), to the left and right of the panels

Typical run

In the Program Code box (see Figure 2) the program (segment) being traced, is displayed. The statements are numbered to facilitate explanation and the drawing of variable diagrams. When a statement is 'executed', it is highlighted in red in the Program Code box and explained in the Explanation box, while the variable diagrams depicting the state of the computer memory at that specific time are reflected in the Memory box and the state of the Console window portrays the output at that stage. Figures 2 to 5 show a representation of the view in each of the four panels for the same statement.

Activity 12.b.i

```
1.  //Activity to investigate the effect of braces
2.  //(curly brackets) on nested if statements:
3.  //Program segment A
4.  if (x > 0)
5.      if (y > x)
6.          cout << "y > x > 0" << endl;
7.      else
8.          cout << "x > 0 and y <= x" << endl;

9.  //Program segment B
10. if (x > 0)
11. {
12.     if (y > x)
13.         cout << "y > x > 0" << endl;
14. }
15. else
16.     cout << "x > 0 and y <= x" << endl;
```

Figure 2: The Program Code box. Line 16 is highlighted because this statement is executed. A scroll bar appears to right because all the program code does not fit into the panel

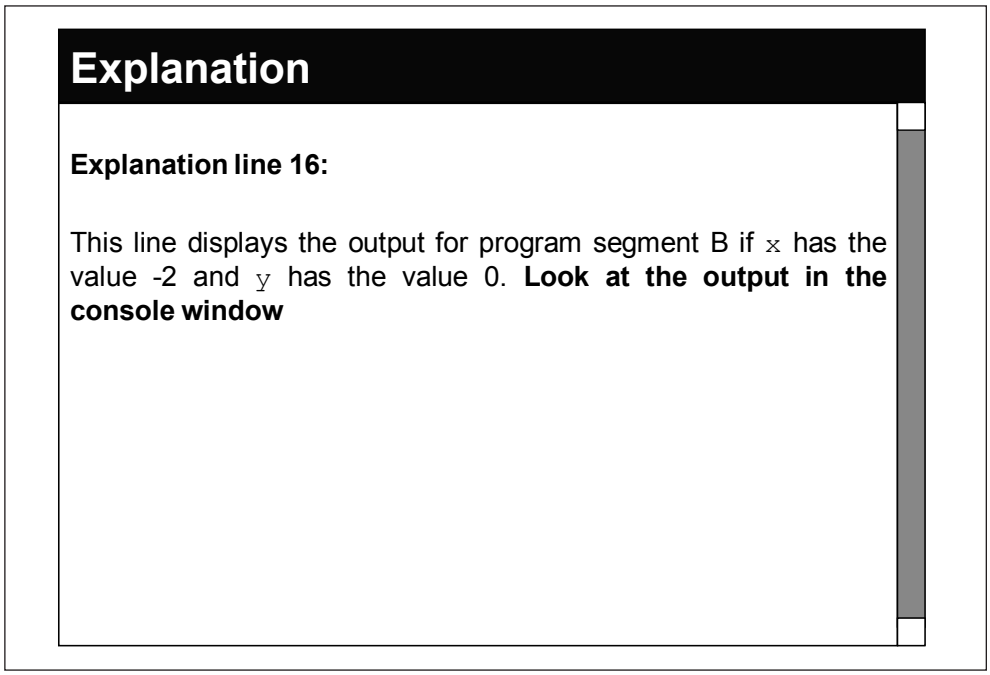

Figure 3: The Explanation box. The prompt to look at the output is displayed in blue

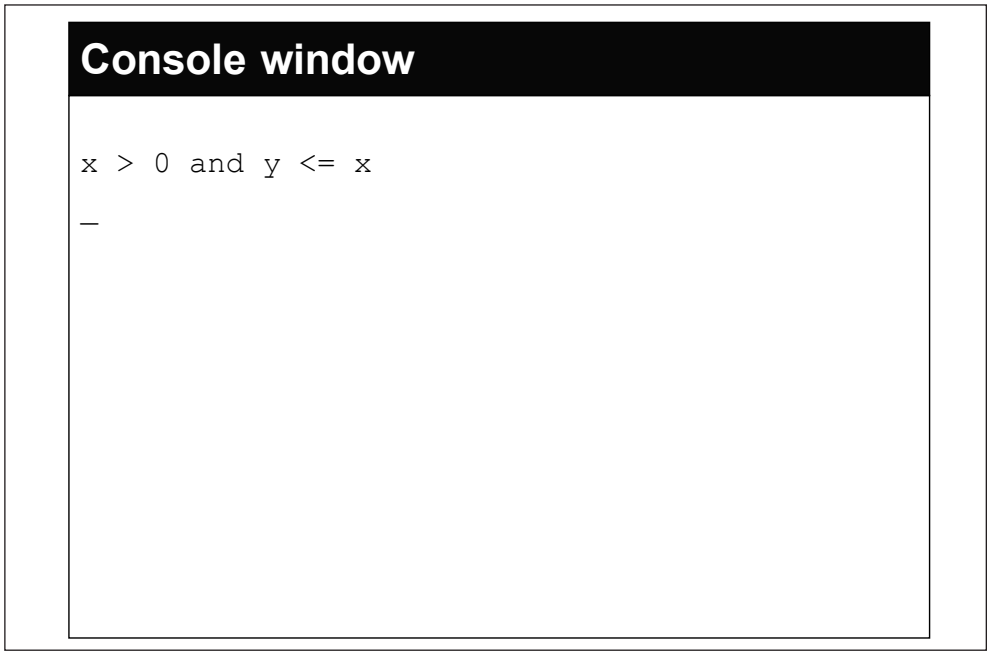

Figure 4: The Console Window box. In the tutorial the console window is black, with white text. The '-' underneath the output, flickers, as it does in the compiler'

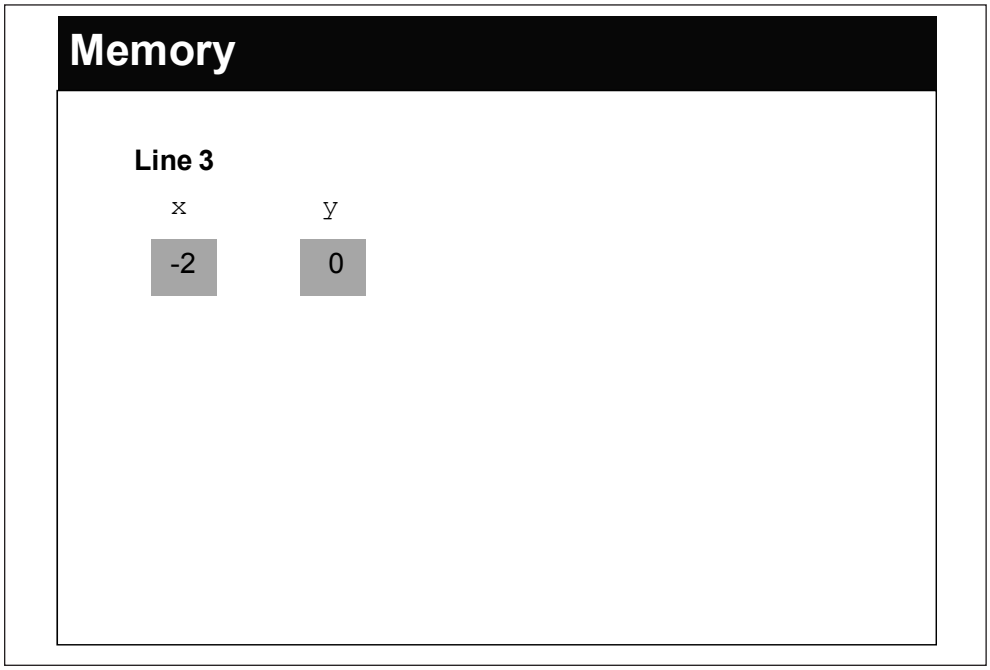

Figure 5: The Memory box. The heading 'Line 3' is displayed in blue. If a variable's value is changed in the statement, the variable diagram box will be circled in red

For example, when the statement in line 16 in the Program Code box in Figure 2 is 'executed', the Explanation box in Figure 3 is also numbered 'Explanation line 16', and gives an explanation of what happens when this statement is executed. Since the statement is an output statement, the Explanation box prompts the user to look at the output produced by this statement in the Console Window box in Figure 4. In the Memory box in Figure 5, the variable diagrams show the contents of the computer memory at this stage. The reason why the Memory box displays variable diagrams for line 3 is because the variable diagrams for variables *x* and *y* have not changed since the initialisation statement in line 3. The user can now confirm that the values that the variables *x* and *y* have in memory are in fact displayed in the output.

Cueing

Cueing is used to guide learners' attention to changes in the Memory box and the Console window. This is a key element that ensures that learning takes place. Cues are given with blue prompts in the Explanation box that instruct them where to look, or red circles in the Memory box to indicate changes in the variable diagrams.

The user is prompted to look at the Console Window box when output is displayed, as well as when input is accepted by the program. The tutorial does not allow the user to input his or her own values, but instead either assumes that specific values will be input and displays these values then in the Console Window box; or lets the user choose between two or three values, in which case the chosen value is displayed in the Console Window box as if the user has keyed it in.

The Explanation box also prompts the user when it is to time to look at the Memory box. When the value of any variable in the Memory box changes, the Explanation box displays a prompt and this change is circled in red in the Memory box to draw attention. Sometimes the user is prompted to look at both the Console Window box and the Memory box, for example when an input statement is 'executed' and an input value for a variable is 'read' from the Console Window and stored in memory. When this happens, the new value for the variable is changed in its variable diagram in the Memory box.

Navigation

Each panel contains a scroll bar that allows the user to see all the codes, diagrams, output or text to see if it does not fit into the panel. Apart from the four panels, the screen also incorporates navigation buttons to allow users to step forward and backward at their own pace through the program trace, and to exit the tutorial or return to the index. Occasionally, the tutorial allows the user to choose between two options during an activity, and then proceed according to this choice.

Design decisions

The tutorial is aimed at novice programmers. It is customised to resemble the static diagrams provided in the study material as closely as possible in order to enhance comprehension thereof, and to enable students to learn how to draw variable diagrams. This correspondence between the tutorial and the related study material is a key factor in its acceptance as a learning tool (Naps et al. 2003; Rössling 2010).

To further align the tutorial with the study material, the Console Window box representing output from the program imitates that of the compiler used by the COS1511 students. The purpose of this is to assist them in developing and reinforcing their own mental models of what happens when a program is executed. The term 'Console window' is also the term used in the compiler used by the students.

The screen layout into four panels, as shown in Figure 1 (one each to show the program code, the computer memory, the program output and an explanation of the step), presents the program simultaneously at these four levels. In this way a

student can keep track of what happens when a statement is executed, see how to draw the variable diagrams indicating the change in memory (if any), read the explanation and also see what output (if any) the statement produces. This allows the student to form a mental model of what happens in computer memory and how to represent and keep track of it by drawing variable diagrams.

To engage students, the tutorial is interactive and allows for direct user control in the form of navigation buttons to move forward or backward through an activity. This allows students to determine the pace, at and direction in which the visualization unfolds.

A user-centred design, with consistency in screen layout and navigation buttons, as well as roll-overs displaying the purpose of a button is followed. A lean, elegant design minimises the cognitive load.

The cues used to guide users' attention are always in the same format. All prompts guiding the user to look at either the Console Window box or the Memory box are displayed in blue, while the circles in the Memory box to draw attention to changes in the variable diagrams are always red. In this way, it becomes 'automatic' knowledge, just enough to nudge the user, as to where to pay attention, without detracting from comprehension.

Changes in variable diagrams in the Memory box are only effected when a program statement that causes a change in the internal memory is executed. Therefore, the Memory box will not display variable diagrams for every variable in each program line. This shows to the student that memory does not change without a program instruction, and it avoids distracting the student by an apparent change in the Memory box, which is in effect only a change in line number.

When demonstrating variable diagrams for activities with loops, both the Explanation box and the Memory box indicate which iteration of the loop is executed. This helps the user to keep track of what happens in memory, and thereby assists in developing the user's mental model of what changes in memory during each step in the loop's execution.

To further engage learners, they are presented with activities at certain points in the presentation. Periodic questions that they have to answer before the tutorial continues, and are also used to keep their attention.

Since Unisa students frequently have limited or intermittent internet access and postal services are unreliable, the tutorial is supplied on CD as part of the study material in a format that allows users to install the tutorial from the CD. To deal with unreliable postal systems, it can also be downloaded from the course website.

Development process and initial testing

The tutorial has been designed by the primary author and developed by a team of three students at the Open Window art academy in Pretoria. The design is based on the way variable diagrams are presented in the study material, combined with the manner in which the compiler used by the students displays output.

For each activity, the designer provided the program code with numbered statements, and the explanations and output (if any) corresponding to each statement, in a Word file to the developers. The program code showed the required indentation. The designer also indicated where the different prompts should appear in the explanation text. She drew variable diagrams for each statement that caused a change in memory by hand, and indicated where the cues to draw attention to changes (red circles) should be shown. The developers then developed the activities in Adobe Flash Player 10 ActiveX.

Initial testing was done by executing each activity and checking statement by statement whether the correct explanation, output and variable diagrams are displayed. Program indentation, prompts and cues were checked. We also checked that the navigation buttons allow the user to proceed backwards and forwards in the correct order, according to program execution.

LESSONS LEARNT AND GUIDELINES DISCOVERED

Initially the developers did not always work at the same venue at the same time, which lead to small inconsistencies in the tutorial. This provided some valuable lessons, of which the first is that, if different programmers are working on different parts of a tutorial, they should work together at the same venue. It led, for example, to three different versions of cueing, as can be seen in Figure 6. This was not necessarily a bad thing, because it allowed us to test for the most effective form of cueing in the HCI lab. Option 1 has a solid red line around the variable diagram, option 2 is circled in red, with the value of the variable displayed in red on a grey background, while option 3 has a flashing red background in the variable diagram box, with the value displayed in black as well as a red circle.

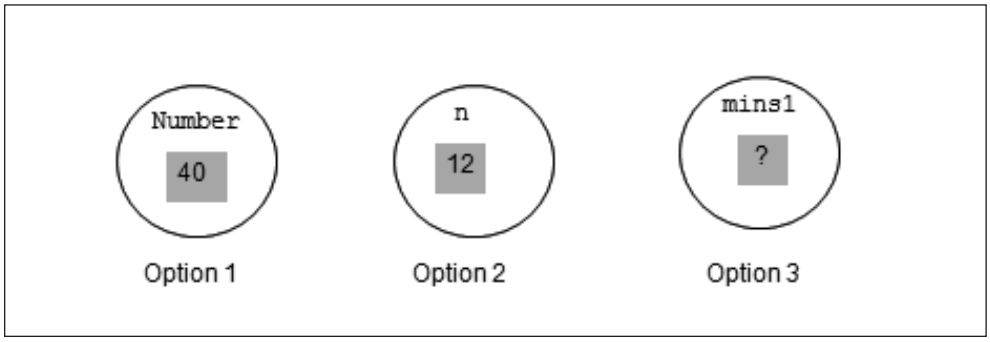

Figure 6: Three different versions of cueing to draw attention to changes in the Memory box

One of the points which was debated by the team offering COS1511 relates to whether variable diagrams for all the previous program lines executed should be shown versus showing only variable diagrams for the current values in memory. Also, as a result of the developers not always working together, both these options were implemented. Looking at the version where only the current values in memory are displayed, it became apparent that this was not advisable. For example, when only the current values of variables are shown, a statement such as

answer = answer + 1,

which changes the value of variable answer, causes the user to backtrack in order to verify what happened.

Another point of debate was whether the variable names in the Memory box should appear above each line or only above each column. Both options were implemented, and while testing the first activities, it became immediately clear that variable names should be displayed above each line in the memory box, since displaying names only above each column forces the user to look upwards to determine the name of a variable after a few lines have been executed. This increases the cognitive load and distracts attention and comprehension. It also makes it impossible to track parameter transfer when functions are executed.

From the development process and initial testing of the tutorial as described here, we identified the following three guidelines for the design of visualizations to enhance comprehension:

1. Deliberately build in elements to help students consolidate their mental model, such as presenting all four panels for a statement simultaneously on the same screen to enable them to integrate the information provided by these four views. Numbering the statements in the Program Code box, the Explanation box and the Memory box also assists with this process, as do labeling the iterations in loops. Showing changes in variable diagrams in the Memory box only when changes in the internal memory occur, and imitating the Console Window of the compiler to show output serves the same purpose.

2. Reduce extraneous cognitive load as much as possible. This is done with the numbering of statements in the Program Code box, the Explanation box and the Memory box, keeping the notation of cues in the Explanation box and the Memory box constant, changing variable diagrams only with changes in the internal memory, and drawing attention to important changes with prompts and cues.

3. Prompt the user judiciously, i.e. neither too much nor too little. For example, using option 3 (in figure 6) to cue users to look at changes in the Memory box, would be overkill, while displaying variable names only above each column instead of above each line in the Memory box, would offer too little information.

For the development process itself, we recommend that the developers of visualizations work together in the same venue to facilitate uniformity in the implementation. They should also consult with the main designer before making design decisions regarding the implementation of the visualization.

FUTURE RESEARCH

The development of the tutorial described in this article, is merely the first step in the effort to improve the throughput in COS1511. It will be followed by evaluating the effect of the students using the tutorial, as indicated in the research methodology. This section gives an overview of the intended research process to allow for evaluation.

The tutorial is supplied as part of the students' study material. To compel students to use the tutorial, the assignments for COS1511 include questions that require the use of the tutorial, as well as a survey on using the tutorial.

The tutorial will also be assessed in the HCI laboratory with a sample of students to evaluate the usability of, and their user experience with the tutorial.

The usability and user experience evaluation use eye tracking to determine what subjects focus on when using the tutorial and what they ignore or miss. Eye tracking and analysing the patterns of visual attention of participants using the tutorial provides knowledge of how learners interact with tutorial. This may identify usability problems with regard to the layout of the tutorial, terminology used, learners' comprehension of the material presented, as well as their navigation through the tutorial.

Once a student has used the tutorial, (s) he will be asked to manually trace a program on the same level of difficulty, followed by a questionnaire. The feedback gathered during the interviews will be combined with the feedback from the survey included with the assignments and evidence from the eye tracking. This will be incorporated in further refinement of the tutorial; for example, to adapt cues in the tutorial to focus the subject's attention on specific elements in the presentation. In this way, the design can be improved, based on learner interaction and feedback.

Examination results for questions on tracing will be collected and compared to similar questions on tracing from previous examinations, where students did not have access to the tutorial, a method similar to the study described in Ma etal. (2008). This will be used to determine whether or not the tutorial had a positive effect on the students' ability to trace. At the same time, it will ensure a true picture of the effect of the tutorial on the programming skills of the students (McCracken etal. 2001), as well as evidence of its usefulness (Sheard et al. 2009).

CONCLUSION

This article described the development process of an interactive tutorial to teach first-year programming students tracing skills, focusing on the needs of learners in an ODL environment. The tutorial itself, as well as the design decisions on which its development was based, was also presented. We drew attention to some lessons learnt and guidelines identified for the design and development of such tutorials. The tutorial is part of a larger research effort to create a framework of guidelines for teaching programming in an ODL environment using visualization.

REFERENCES

Adamchik, V. and A. Gunawardena. 2003. *A learning objects approach to teaching programming*. Paper presented at the Information Technology: Coding and Computing (Computers and Communications) ITCC 2003, 28--30 April 2003.

------. 2005. *Adaptive book: Teaching and learning environment for programming education.* Paper presented at the Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on, 4--6 April 2005.

Ben-Ari, M., R. Bednarik, R. Ben-Bassat Levy, G. Ebel, A. s. Moreno, N. Myller and E. Sutinen. 2011. A decade of research and development on program animation: The Jeliot experience. *Journal of Visual Languages & Computing* 22(5): 375--84.

Bornat, R., S. Dehnadi and Simon. 2008. *Mental models, consistency and programming aptitude.* Tenth Australasian Computing Education Conference (ACE 2008).

Bower, M. 2009. Discourse analysis of teaching computing online. *Computer Science Education* 19(2): 69--92.

Bruce-Lockhart, M. P. and T. S. Norvell. 2007. *Developing mental models of computer programming interactively via the web*. 37th ASEE/IEEE Frontiers in Education Conference, Milwaukee, WI.

Cavus, N., H. Uzunboylu and D. Ibrahim. 2007. Assessing the success rate of students using a learning management system together with a collaborative tool in web-based teaching of programming languages. *Journal of Educational Computing Research* 36(3): 301--21.

Čisar, S. M., D. Radosav, R. Pinter and P. Čisar. 2011. Effectiveness of program visualization in learning Java: A case study with Jeliot 3. *International Journal of Computers, Communications & Control* VI (2011) no. 4 (December): 669--82.

Clear, T., A. Philpott, P. Robbins and Simon. 2009/2010 Dec/Jan. Report on the Eight BRACElet Workshop: BRACElet Technical Report 01/08 AUT University, Auckland. *Bulletin of Applied Computing and Information Technology* 5(1): 13.

Clear, T., J. Whalley, P. Robbins, A. Philpott, A. Eckerdal, M. Laakso and R. Lister. 2011. Report on the final BRACElet workshop: Auckland University of Technology, September 2010. *Journal of Applied Computing and Information Technology & People* 15(1).

Cogan, E. and C. Gurwitz. 2009. Memory tracing. *Journal of Computer Science* 5(8): 608--613.

Corney, M. 2009. Designing for engagement: Building IT systems. ALTC First Year Experience Curriculum Design Symposium 2009, Queensland, Australia: QUT Department of Teaching and Learning Support Services.

Dale, N. B. 2006. Most difficult topics in CS1: Results of an online survey of educators. *SIGCSE Bulletin* 38(2): 49--53.

Domingue, J. B. and P. Mulholland. 2007. Teaching programming at a distance: The Internet Software Visualization Laboratory. *Journal of Interactive Media in Education (JIME)* 1: 1--31.

El-Sheikh, E. M., J. W. Coffey and L. J. White. 2008. Exploring technologies, materials, and methods for an online Foundational Programming Course. *Informatics in Education* 7(2): 259--276.

Foutsitzis, C. and S. Demetriadis. 2008. AlCoLab: Architecture of Algorithm Visualization System. Advanced Learning Technologies, 2008. ICALT '08. Eighth IEEE International Conference, Santander, Cantabria.

Hevner, A. and S. Chatterjee. 2010. *Design research in Information Systems Theory and Practice.* New York Dordrecht Heidelberg London, Springer.

Isohanni, E. and M. Knobelsdorf. 2010.Behind the curtain: Students' use of Vip after class. In *Proceedings of the Sixth International workshop on Computing Education research*. Aarhus, Denmark: ACM.

------. 2011. Students' long-term engagement with the visualization tool Vip. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, 33--38. Koli, Finland: ACM.

Kaila, E., T. Rajala, M.-J. Laakso and T. Salakoski. 2009. Effects, experiences and feedback from studies of a program visualization tool. *Informatics in Education* 8(1): 17--34.

------. 2010. Effects of course-long use of a program visualization tool. In *Proceedings of the Twelfth Australasian Conference on Computing Education -- Volume 103*, 97--106. Brisbane, Australia: Australian Computer Society, Inc.

Kinnunen, P., R. McCartney, L. Murphy and L. Thomas. 2007. Through the eyes of instructors: A phenomenographic investigation of student success. *Proceedings of the third international workshop on Computing education research*. Atlanta, Georgia, USA, ACM.

Koifman, I., I. Shimshoni and A. Tal. 2008. MAVIS: A multi-level algorithm visualization sytem within a collaborative distance learning environment. *Journal of Visual Languages and Computing* 19(2008): 182--202.

Lahtinen, E., K. Ala-Mutka and H.-M. Järvinen. 2005. *A study of the difficulties of novice programmers*. ITiCSE'05, Monte de Caparica, Portugal, ACM.

Levy, R. B.-B. and M. Ben-Ari. 2007. We work so hard and they don't use it: Acceptance of software tools by teachers. *ACM SIGCSE Bulletin* 39(3): 246--250.

Lister, R., E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon and L. Thomas. 2004. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin* 36(4): 119--150.

Lister, R., T. Clear, Simon, D. J., Bouvier, P., Carter, A., Eckerdal, J., Jacková, M. Lopez, R., McCartney, P. Robbins, O. Seppälä and E. Thompson. 2009. *Naturally occurring data as research instrument: Analyzing examination responses to study the novice programmer*. ITiCSE'09, Paris, France, ACM.

Lister, R., C. Fidge and D. Teague. 2009. Further evidence of a relationship between explaining, tracing and writing skills in Introductory Programming. *Proceedings of the 14th annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*. Paris, France, ACM.

Lopez, M., J. Whalley, P. Robbins and R. Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. *Proceeding of the Fourth international Workshop on Computing Education Research*. Sydney, Australia, ACM.

Ma, L., J. Ferguson, M. Roper, I. Ross and M. Wood. 2008. Using cognitive conflict and visualisation to improve mental models held by novice programmers. *ACM SIGCSE Bulletin* 40(1): 342--346.

MacDonald, M., B. Dorn and G. MacDonald. 2004. *A statistical analysis of student performance in online computer science courses.* SIGCSE'04, Norfolk, Virginia, USA, ACM.

Malan, D. J. 2009. *Virtualizing office hours in CS 50*. ITiCSE'09, Paris, France, ACM.

McCracken, M., V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting and T. Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin* 33(4): 125--140.

Meisalo, V., E. Sutinen and S. Torvinen. 2003. Choosing appropriate methods for evaluating and improving the learning process in distance education programming courses. 33rd ASEE/IEEE Frontiers in Education Conference. Boulder, CO, USA, IEEE.

Meiselwitz, G. 2002. Using the Web to maintain the benefits of small class instruction in large classes. *Journal of Computing Sciences in Colleges* 2011.

Milne, I. and G. Rowe. 2002. Difficulties in learning and teaching programming -- Views of students and tutors. *Education and Information Technologies* 7(1): 55--66.

Mselle, L. J. 2010. *Enhancing comprehension by using random Access Memory (Ram) diagrams in teaching: Class experiment.* Paper presented at the 22nd Annual Psychology of Programming Interest Group, Universidad Carlos III de Madrid, Leganés, Spain, 19--22 September 2010.

Murphy, C., D. Phung and G. Kaiser. 2008. A distance learning approach to teaching eXtreme Programming. ITiCSE'08, Madrid, Spain, ACM.

Naps, T. L., G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. A. Velazquez-Iturbide. 2003. Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin* 35(2): 131--152.

Ng, S., S. Choy, R. Kwan, S. Chan, R. Lau, Q. Li, R. Cheung and W. Liu. 2005. A web-based environment to improve teaching and learning of Computer Programming in distance education. ICWL 2005. 727: Springer Berlin / Heidelberg.

Oates, B. J. 2006. *Researching information systems and computing*. London, SAGE Publications Ltd.

Ragan, L. 2003. Defining quality on your own terms. In *Distance Education Report* 7(3): 3--6.

Rajala, T., M.-J. Laakso, E. Kaila, and T. Salakoski. 2007. *Ville -- a Language-Independent Program Visualization Tool.* Paper presented at the Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007), Koli National Park, Finland, November 15--18, 2007.

------. 2008. Effectiveness of program visualization: A case study with the Ville Tool. *Journal of Information Technology Education: Innovations in Practice* 7.

Reeves, P., Baxter and C. Jordan. 2002. Teaching computing courses -- computer literacy, business microcomputer applications, and introduction to programming online utilizing webCT. *J. Comput. Small Coll* 18(1): 290--300.

Robbins, S. 2008. A three pronged approach to teaching undergraduate operating systems. *SIGOPS Oper. Syst. Rev.* 42, no 6 (2008): 93--100.

Robins, A. 2010. Learning edge momentum: A new account of outcomes in CS1. Computer Science Education 20(1): 37 -- 71.

Rössling, G. 2010. A family of tools for supporting the learning of Programming. *Algorithms* 2010(3): 168--182.

Shaffer, C. A., M. Cooper and S. H. Edwards. 2007. Algorithm visualization: A report on the state of the field. *ACM SIGCSE Bulletin* 39(1): 150--154.

Sheard, J., S. Simon, M. Hamilton and J. Lonnberg. 2009. Analysis of research into the teaching and learning of programming. *Proceedings of the fifth international workshop on Computing education research workshop*. Berkeley, CA, USA, ACM.

Simon, S., A. Carbone, M. d. Raadt, R. Lister, M. Hamilton and J. Sheard. 2008. Classifying computing education papers: Process and results. *Proceeding of the Fourth international Workshop on Computing Education Research*. Sydney, Australia, ACM.

Thomas, L., M. Ratcliffe and B. Thomasson. 2004. Scaffolding with object diagrams in first year Programming classes: Some unexpected results. *SIGCSE Bulletin* 36(1): 250--254.

Tsai, C.-W. 2012. An effective online teaching method: The combination of collaborative learning with initiation and self-regulation learning with feedback. *Behaviour & Information Technology*: 1--12.

Tsai, C.-W., P.-D. Shen and M.-C. Tsai. 2011. Developing an appropriate design of blended learning with web-enabled self-regulated learning to enhance students' learning and thoughts regarding online learning. *Behaviour & Information Technology* 30(2): 261--71.

Ury, G. 2005. A longitudinal study comparing undergraduate student performance in traditional courses to the performance in online course delivery. *The Information Systems Education Journal* 3(20): 3--9.

Urquiza-Fuentes, J. and J. Á. Velázquez-Iturbide. 2009. A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Transactions on Computing Education* 9(2): 1--21

Vaishnavi, V. and W. Keuchler. 2004. Design research in information systems. Available at: www.isworld.org/Researchdesign/drisISworld.htm (accessed ????).

Venables, A., G. Tan and R. Lister. 2009. A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the fifth international workshop on Computing education research workshop*. Berkeley, CA, USA, ACM.

Virtanen, A. T., E. Lahtinen and H.-M. Järvinen. 2005. *Vip, a visual interpreter for learning Introductory Programming with C++*. Paper presented at the Koli Calling 2005 Conference on Computer Science Education, Koli, Finland.

Whalley, J. L. and R. Lister. 2009. The BRACElet 2009.1 (Wellington) Specification. Eleventh Australasian Computing Education Conference (ACE2009), Wellington, New Zealand, Australian Computer Society, Inc.