LAYOUT FOR LEARNING
Designing an Interface for Students Learning to Program

Suzan Badri
Academic Development Centre, Kingston University, London, U.K.

James Denholm-Price, James Orwell
Faculty of Computing, Information Systems and Mathematics, Kingston University, London, U.K.

Keywords: Content delivery, Presentation, Programming.

Abstract: Many students have difficulty learning to program. It is conjectured that this difficulty may be increased by a disorganisation of the resources available to the student while they are learning. The unfamiliarity of terms and concepts, and frustration with mysterious errors, is exacerbated by the struggle with multiple windows and the attempt to memorize patterns which would be better viewed concurrently. As a consequence, a layout for learning programming is proposed: the aim of the proposal is to ensure that the students can easily arrange for the relevant resources to be displayed concurrently, without further manipulation of the application windows. Three types of resource are considered: the editor, the question sheet (instructions) and further reference resources such as glossaries, descriptions of concepts and common tasks. An HTML template is proposed to accommodate these last two types of resource. It is designed to allow all three materials to be positioned and selected and thereby allow for the concurrent display of the relevant resources. An evaluation of these proposals is presented, and the prospects for further development are considered.

1 INTRODUCTION

Over the past decade, on-line learning materials have become commonplace – especially for Information Technology subjects such as Computer Programming. This ‘E-learning’ material has facilitated worldwide interaction and the exchange of information between field experts and learners. It created instant access to global resources, increased communication, and enabled the possibility to share and publish information to a global audience (Dabbagh, 2005).

The effectiveness of the e-Learning material depends on many factors. These include: the relevance and quality of the material; the extent of support from peers and instructors; and the fitness of the IT hardware used for this purpose. The format and layout of the e-learning material is also an important factor. This is especially true in a subject area such as Computer Programming, which students often find bewildering and frustrating. A short review of previous work in this area is provided in Section 2.

E-learning resources are usually the product of a process that involves developing content, storing and managing content, packaging content, student support and assessment (Govindasamy, 2001). This paper proposes a simple layout styling for instructions and reference material. The design considerations are discussed in Section 3.The focus is: how the material is presented to the student, and how the interaction is planned. A simple html implementation is presented in Section 4. The emphasis is on straightforward, easily packaged material, without requirements for server-side functions, since these may inhibit the applicability and sustainability of the material. The on-going evaluation is described in Section 5. A discussion of further work and some concluding remarks are provided in Section 6.

2 PREVIOUS WORK

2.1 e-Learning Theories and Systems

Developers of e-learning resources should consider the usability and accessibility of the resources as a priority. These considerations may reduce or prevent the digital divide' (Ardito et al., 2006). The digital divide is a concept that describes the difference between those who have access to on-line resources and those who have not; or between users and non-users. The digital inequality is an extension to describe the inequality in having tools, equipments, resources, support, skills, and cognitive abilities for those who are already on-line (DiMaggio and Hargittai, 2001).

One framework for e-learning, proposed by Dabbagh (Dabbagh, 2005), proposed three components: Pedagogical models, Instructional and learning strategies and Learning technologies. Pedagogical models are cognitive models or theoretical structures originated from the knowledge about cognitive process that helps the learner to encode information into their memory. They are based on learning theory and they provide a link from theory to practice. The instructional and learning strategies are derived from the Pedagogical models: the instructor and the designer can use these strategies to plan how the learner can engage with learning process. The result of translating a pedagogical model into a strategy can result into a number of activates for the learner; the learning technologies are the tools to present a case or a problem using graphics, video, audio, animation and hypertext (Dabbagh, 2005).

The aim of developing an educational tool should be to help students to learn, therefore it is essential that the tool must be engaging, support the learning outcome, intuitive and natural (Ardito et al., 2006). The way in which the tasks are presented to the learner should follow effective usability guidelines, and effective evaluation methodologies to implement usable interfaces. Often, e-learning systems are electronic transcript of the traditional material could be presented through unfriendly system, confusing menu, many clicks, different screens, unrelated tags. This may reduce the learner's motivation and attitude. Arguably, much of students' activity within Virtual Learning Environment is administrative, such as reading announcements and downloading lecture notes.

The success of e-learning resources depends on the level of its usability: if the system isn't usable, students can be distracted, and instead of trying to learn how to use it, they only use it to obtain content. As students are learning new concepts, they need an easy-to-use system with updated content. Staff, on the other hand, will be looking for a system that they can keep the content updated (Ardito et al., 2006).

2.2 Challenges Specific to Learning Programming

The activity of *learning* to programme is itself a skill (Wolf, 2003). Learning a programming language requires familiarity with a set of skills and processes (Jenkins, 2002). Jenkins describes that the hierarchy of skills starts with learning the basics of the syntax, semantics, structure which leads to learning a specific style. A typical approach for programmers is as follows: the process starts with a specification; this is translated into an algorithm, the algorithm is mapped to a set of instructions based on previous experience and then finally the algorithm is translated into program code. The process of learning to programme has been described to be 'systematic, incremental and non-linear'. Beginners are encouraged to start with small lines of code, test them, and sometimes re-think the prior structure of the programme (Bennedsen and Caspersen, 2005).

First-year university students are usually taught programming by a weekly lecture and practical session. The lectures introduce them to the theory which covers concepts, program syntax and methods. The students are required to complete practical sessions which require learning a combination of skills: the structure and syntax of the programming language, the integrated development environment (IDE) and the development process. Students must also solve a series of questions and this is usually is part of the module assessment.

In the practical sessions, students must simultaneously marshal the necessary on-line documents and editors, switching their attention between them as necessary. They are faced with a limited screen size and they are typically novice users of the tools. The user may become lost and confused while trying to find the relevant information (Lif et al., 2001). Some students may face what is known as 'cognitive overload' (Mayer and Moreno, 2003): a term used to describe what learners, in particular beginners, can face when asked to use multiple systems. An increase in learners' cognitive load may result in a decrease in students' motivation to learn (Wolf, 2003).

Jenkins discusses 'cognitive factors' (Jenkins, 2002): the motivation to learn is one of these factors that Jenkins considers may explain why students find programming difficult. He suggests that some students have natural motivation to learn programming, while some have a motivation to succeed and endure a successful career and other are pressured by their social surroundings and families.

It has been suggested by Hodges that students who are motivated to learn could have a better chance to succeed compared with those who are not. Students who are learning well will have a higher level of motivation to learn in the future (Hodges, 2004).

Jenkins also suggests that students' learning style is another cognitive factor in the difficulties found by students to learn to program. However, with a traditional classroom one teacher to many students, it's difficult to have an individualised learning style (Wolf, 2003). The creators of the iweaver' project have designed a tool to accommodate individual learning styles in an adaptive e-learning environment that teaches the Java programming language. The project was an e-learning resource with dynamically adapted digital content might be an effective method for personalised learning.

2.3 Existing Approaches

Here, the e-learning resources and specific projects used in computer programming will be discussed. The dynamic approach in teaching programming is to use or to integrate e-learning resources. Recordings of teaching materials is one approach, this can be achieved either by setting a lecture theatre for recording the teacher while they demonstrate the solutions, this will involve the use on an IDE, on-line documentation, testing and debugging. These recording could also be streamed.

2.3.1 Videos of Screen Captures

These are pre-recorded tutorials and solutions to programming problems using screen capture software. They are known as 'process recording' (Bennedsen and Caspersen, 2005) or Screencasts (Lee et al., 2008). The teacher talks through the problem and the student can follow on their own pace and view the record over a number of times. This type of resources has been proved to be popular and successful with students and teachers (Bennedsen and Caspersen, 2005). There is a number of recording software, such as Camtesia, Captivate and Window Media Encoder and Editor, there is no need to extra software or a special studio to create these tutorials. The videos can be delivered independently or embedded in a web-page and delivered within a Virtual Learning Environment.

2.3.2 i weaver

The' i weaver' project was developed to highlight the importance of students' different learning styles in an adaptive e-learning environment that teaches Java programming language (Wolf, 2003). This project used an inventory to assess the students' learning styles, it used a 118 multiple choice question based on the Dunn & Dunn learning style model. Before entering the learning environment, the students receive an analysis of their learning style profile. When the students enter the learning environment they presented with two out of four media experience (Visual text, visual pictures, interactive animations and auditory media experience.) This is to decrease the cognitive overload (Wolf, 2003). The learning content are divided into modules, the modules have units, learners can repeat a unit using a different media experience or move to the next unit. The learner is asked to rate their experience and overall satisfaction. The set of multiple choices tests are also conducted after each module.

2.3.3 BlueJ

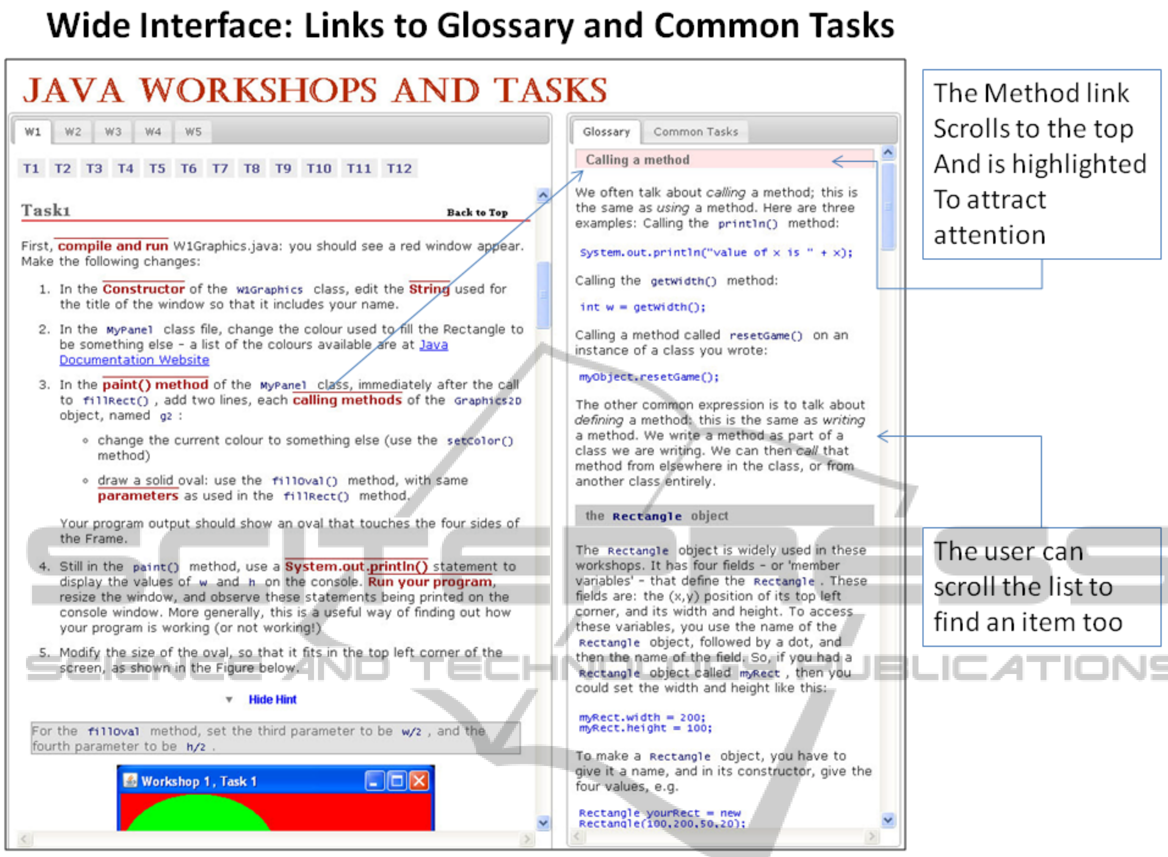
The BlueJ software is an integrated development environment. It is designed to teach introductory Java programming language using a set of resources, examples and resources based on a pedagogical approach. The creators of BlueJ aimed to make the environment an object oriented through using visual methods that help to represent objects and classes and the relationship between them. They demonstrate that their method of visualisation and the frequent interaction with single objects and methods can give the students a better understating of the Java language. They also claim that BlueJ has a simple user interface with minimal distractions. This can also help to increase students' understanding of the programming principles (Kölling, 2008)

3 DESIGN CONSIDERATIONS

The point of departure, for designing the on-screen layout of workshop material, is that the student should easily be able to arrange for all relevant information to be concurrently displayed. We consider there to be three main elements of this material:

1. The set of workshop questions, possibly including snippets of source code or starting programs.
2. Reference material, for example glossaries, 'lecture note'-style descriptions of concepts, and instructions how to complete common tasks.
3. The 'development environment': this could be a text editor with compile command or command line, or an 'integrated development environment'

Using previous observations of students' interaction with learning material, the following issues were noted:


Wide Interface: Links to Glossary and Common Tasks

JAVA WORKSHOPS AND TASKS

W1 W2 W3 W4 W5

T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12

Task1 Back to Top

First, compile and run W1Graphics.java: you should see a red window appear. Make the following changes:

1. In the Constructor of the `W1Graphics` class, edit the String used for the title of the window so that it includes your name.
2. In the `MyPanel` class file, change the colour used to fill the Rectangle to be something else - a list of the colours available are at Java Documentation Website
3. In the paint() method of the `MyPanel` class, immediately after the call to `fillRect()`, add two lines, each calling methods of the `Graphics2D` object, named `g2`:
   - change the current colour to something else (use the `setColor()` method)
   - draw a solid oval: use the `fillOval()` method, with same parameters as used in the `fillRect()` method.

   Your program output should show an oval that touches the four sides of the Frame.
4. Still in the `paint()` method, use a System.out.println() statement to display the values of `w` and `h` on the console. Run your program, resize the window, and observe these statements being printed on the console window. More generally, this is a useful way of finding out how your program is working (or not working!)
5. Modify the size of the oval, so that it fits in the top left corner of the screen, as shown in the Figure below.

Hide Hint

For the `fillOval` method, set the third parameter to be `w/2`, and the fourth parameter to be `h/2`.

Workshop 1, Task 1

Glossary Common Tasks

Calling a method

We often talk about *calling* a method; this is the same as *using* a method. Here are three examples: Calling the `println()` method:

```
System.out.println("value of x is " + x);
```

Calling the `getWidth()` method:

```
int w = getWidth();
```

Calling a method called `resetGame()` on an instance of a class you wrote:

```
myObject.resetGame();
```

The other common expression is to talk about *defining* a method: this is the same as *writing* a method. We write a method as part of a class we are writing. We can then *call* that method from elsewhere in the class, or from another class entirely.

the `Rectangle` object

The `Rectangle` object is widely used in these workshops. It has four fields - or 'member variables' - that define the `Rectangle`. These fields are: the (x,y) position of its top left corner, and its width and height. To access these variables, you use the name of the `Rectangle` object, followed by a dot, and then the name of the field. So, if you had a `Rectangle` object called `myRect`, then you could set the width and height like this:

```
myRect.width = 200;
myRect.height = 100;
```

To make a `Rectangle` object, you have to give it a name, and in its constructor, give the four values, e.g.

```
Rectangle yourRect = new
Rectangle(100,200,50,20);
```

Figure 1: The proposed two-column interface with some example content. Left: Workshop instructions (tasks) with hints that can be revealed. Right: two tabs of content containing a glossary of terms, and descriptions of common tasks and concepts.

1. Many students would have difficulty in locating the relevant supporting material. For example, they would not always locate the exact lecture slides that provide the help that they need (nor recognize those slides if they had located them). Or they would use an internet search, and then use retrieved material that was not always appropriate (such as integrating some more advanced techniques that they had not yet covered).
2. Most students would sometimes have difficulty in organising all the material that they had located. For example, relatively few students would attempt to organise the windows side-by-side; instead they alternate their attention between full screen windows.
3. Many students would seem to encounter less difficulties in learning computer programming if the task of locating the relevant information, and arranging it on-screen, were easier.
4. A programming task typically has a series of components which must be competed in order. Determining the amount of detail that should be presented to the student is a difficult balance. Too much detail, and the overall objective becomes obscured and the student becomes reliant on word-by-word instructions. Too little detail, and students not knowing how to start the task are left without any clear route.

A general point about the on-screen layout of material can be made. The aspect ratio of computer screens is 'landscape', and increasingly so: from 4:3 to 16:9, with HD resolution of 1920 by 1080. This is useful for arrangement of multiple columns of content, side by side. We plan for the three elements of material listed to each be displayed in a separate column of the student's display. The 'development environment' is a separate (single column) application window, occupying one-third of the display width. However, there are advantages for integrating the workshop questions, and reference material, into a single web browser window, having two columns and occupying two-thirds of the display.

These advantages are several-fold. Firstly, as a single window on the student's display, it is easier for the student to organise and in particular trivial to arrange to that both questions and reference material are visible. Secondly, hyperlinks can be used to the Workshop content area to the relevant parts of reference material, which would be displayed concurrently in the reference content area. Finally, there is arguably an advantage in making an explicit requirement on the content author, to provide adequate reference material.

Thus, the first design decision is a two-column organisation of the content, with question-specific material on the left and general reference material on the right. The second design decision is to provide a facility for detail or hints about a particular task to be initially hidden, and only revealed when requested by the student. This allows the overall flow of the tasks to be viewed, and also encourages students to attempt the questions before revealing the hint and using the source code it provides.

The design of the interface will reflect the learning methods for programming which is a combination of a surface' and 'deep' methods (Jenkins, 2002). The surface learning style can help to memorise the syntax of the language; the deep learning style gives the learner the ability to spot a problem and find a solution based on previous experiences. The design of the interface will also consider the use of multimedia.

The interface will present some information simultaneously and take into account the relationship between the presentation of information and decision making and how results from research indicates that simultaneous presentations of information can lead to a faster decision making (Lind, 1994).

The other important step is to provide accessible e-learning resources to a wider audience, users with different cultural background, cognitive capabilities, and technical skills. Successful resources will generate meaningful learning: this term is used to define the interaction between pedagogical models, instructional strategies, and learning technologies (Dabbagh, 2005).

3.1 Examples of Similar Designs

Graphical User Interface Applications have several conventions for organising areas of content. Perhaps the most sophisticated is the 'Integrated Development Environment' (IDE), used by developers of computer programs and html content. These environments provide multiple views of the same content and also provide links between different content areas. Access to documentation and associated literature is also included in the design. One issue here is that familiarity with this environment itself takes some time and for the novice programmer this aspect alone can be a significant barrier (Kölling, 2008).

For the more general user, the family of document preparation applications (such as Microsoft Word, Powerpoint) provide some more conventions for multiple content areas. These can use a 'summary' and 'detail' area for rapid navigation around a single large document (or set of slides). There is also facility for comparing different versions of the same document side by side. With some applications, the 'help' facility explicitly resizes the original window to ensure that the user can view this material alongside the existing window.

In html documents, there are some increasingly established conventions for the organisation of content, e.g. internal navigation along the top and the left of the page, with external and related links in a right side bar. The use of two column format for a single document is discouraged, because the scrollbar control makes it unweildy to navigate. Only one example was found of two column format used for two documents in the proposed manner (http://www.xml.com/axml/testaxml.htm). Here, however, the material in the right 'reference' column was a collection of footnotes. In the proposed approach, the reference material is a more organised body of content that can be read through and browsed independently.This design, together with indicative content, is shown in Fig. 1.

4 IMPLEMENTATION

4.1 Content Structure

The proposed design encourages the student to complete the programming tasks using two window browsers, the interactive workshops window and the programming editor window. This simultaneous way of organisation of windows may encourage the student to focus on learning and completing the tasks with to learn about the topic and complete the tasks by limiting the destruction of multiple open windows.

The use of Frameset technology was considered and ultimately rejected for this project. This is because the status of Framesets uncertain in the HTML 5 standard, there are issues regarding accessibility and visibility of the resources.

The interface has two versions: a 'Wide Interface' and a 'Small Interface'. The Wide Interface is designed to give the student maximum view of content and navigation. The Small Interface is designed for smaller screens and other mobile devices such as the 'ipad'. The Small Interface is presents the same content but with a different layout and naviagation.

In the design for a wide screen, the interface will be presented as a webpage with two column structure. The left side column will contain a list of tasks for each workshop. Each task could have hints' that would help the student to complete the task. The default status for the hint will be closed, this is to give the student the option to view the hint or not.

The right column of the interface contain a tab menu, the menu has two items, the Glossary and the Common tasks. The glossary items are terminologies that the student can refer to, such as, the definition of a Constructor or the definition of a Method. The Common Tasks, are a set of instructions that are common to the process of solving tasks, for example, How to Create a Constructor' is a common task that a number of tasks will require creating a constructor.

4.2 Styling and Interaction

Both columns have independent scrollbars, the user can scroll down the tasks column without effecting the Glossary and Common Tasks column. The links from the tasks column target the appropriate tab and anchors the correct item, a distinctive animation and background colour is used to attract the user attention, the animation are colour fades in a settle motion, this way the user is aware of what they have clicked.

In the design for the small screen display, the interface is a one column web-page. The tasks are displayed with the same style as the Wide Layout. The Glossary items and the Common Tasks pop out using a hyper link. Or the user can scroll down the page and locate an item.

The transition between the two styles is determined by the use of the appropriate Cascading Style Sheet using Media Query, this allows the layout to be changed to suit the screen or a device without changing the content.

The interface design has separate content from style approach. The content are tagged and styled using HTML and CSS, to update or edit the content, the teacher can use any HTML editor. All the defined styles are stored in the CSS file, the Jquery code is stored in a separate file.

5 EVALUATION STRATEGY

A usability study was an essential component of the development of the Wide and the Small Interfaces. The goals for this study were: to establish which Interface is preferred by users; and to highlight the design issues in order to improve efficiency, user satisfaction and the effectiveness of the Interface as a learning tool. During the development of the Interface, several evaluation methods were being used. Firstly, a Heuristic Evaluation' (Nielsen, 1994) was carried out during the iterative design process. This helped to identify some elementary usability problems which were fixed during the design and implementation phase. Secondly, a Systematic Usability Evaluation' methodology was employed. This combines a rigorous functional inspection with user-based evaluation. Some studies (Ardito et al., 2006) have outlined that these methods are complementary and they can be combined for obtaining a reliable evaluation process. Thirdly, the Wide and the Small Interfaces were evaluated using an Inspection Method', using usability guidelines and requirements to test the actual Interface. At first, simple test content was used, moving on to indicative content when appropriate. This method was designed to measure learn-ability, ease of use, efficiency, helpfulness, satisfaction and positive attitude towards programming.

A User-based Study was organised: students were invited to participate in the study. The students were asked to complete a number of programming tasks using three Interfaces. These three Interfaces were as follows:

- The existing resources (Microsoft Word and PowerPoint documents accessed via Blackboard)
- The Wide Interface
- The Small Interface.

Their performance was captured using the Morae' software (Techsmith, 2010). This software can be used to observe the user while they attempt to complete the tasks: to capture their interaction during the usability test and to record their comments. The software can also be used to design questionnaires, release them when the user completes a task and store all the gathered results. The results can be analyzed and presented using a report template generated by the software. All the participants were asked to verbalise their thoughts and decisions while they complete the tasks. They were asked to complete three sets of questionnaires, one before they start the tasks, one after each task and a final set of questions and an interview at the end of the test. This gave them the opportunity to express which Interface they preferred and give recommendations for improvements. The observation during the study helped to collect data to measure the participants' task success rate, time taken to complete the task using each Interface and to record help, assistance or errors made by the participant.

The gathered data from the usability study were

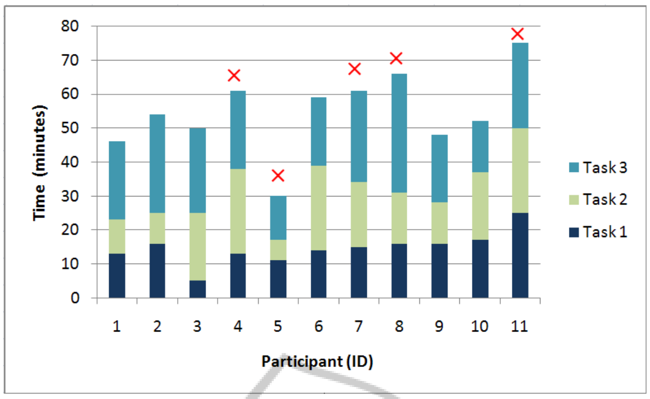


Figure 2: Time on Task is shown for each participant using different colours; the participants who didn't complete Task 3 are shown with an x'.

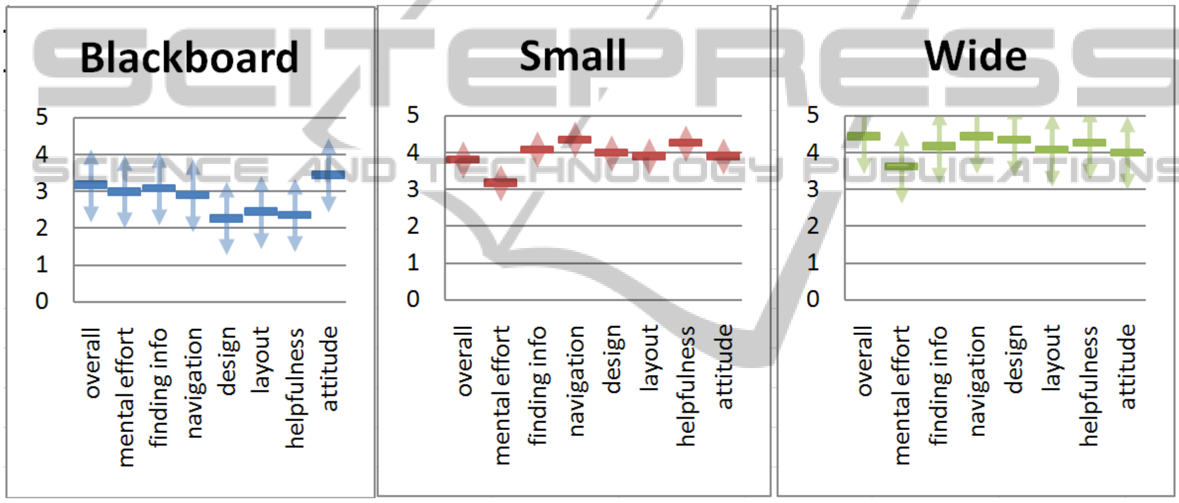


Figure 3: Arrows indicating one standard deviation either side of the mean response from all participants.

organised, analysed and presented into categories that reflected the requirements and the actual usability study and some additional area based on the participants' experience. The raw data collected from the surveys were based on rating values given to a certain criteria, such as the mental effort or the ease of use for each Interface. These data were used to make a graphical representation of the results. Another type of data was collected through observing the participants, interviews after each task and the feedback that each participant gave at the end of the study.The participants will be referred to as P' with their allocated number.

One example of the performance indicators measured in the trials, was the Time on Task' and Task Success'. The participants' ability to complete the tasks on time varied considerably. They all managed to complete Task 1 and Task 2 within (or nearly within) the allocated time. The allocated time was 20 minutes: extra time of 5 minutes was given when needed. Task 3 took longer, and it wasn't completed by five participants as shown in Fig. 2 (the red x indicated the participants who didn't complete Task 3).

Fig. 3 shows the mean response (over all 11 participants) from the questionnaire, for each of the categories listed above, for each of the three Interfaces. We can see that, in all categories, the same order of preference is maintained. The Wide Interface is most preferred, followed by the Small Interface. The Blackboard Interface is least preferred. In the following Section, we consider whether this preference is statically significant.

5.1 Statistical Significance Analysis

The responses from participants indicated some preference towards the Wide and Small Interfaces (in that order). Some further analysis can be undertaken to es-

Table 1: Significance tests for comparison of the three Interfaces.

	t	1-tail	2-tail
Bb vs Small	1.606	6.2%	12.4%
Bb vs Wide	4.146	0.025%	0.05%
Small vs Wide	1.898	3.6%	7.2%

tablish whether the participants' responses indicates a significantly different attitude to the three different Interfaces. To perform this significance test', we will estimate the probability that the mean responses would differ by at least as much as was actually observed, given the Null Hypothesis', *i.e.* that all the responses were really drawn from the same distribution, and the observed differences were just down to random variations.

The number of participants (11) is quite small for this type of analysis. For this reason we do not attempt to look for significant differences in the variance. So the alternative hypothesis is that the different Interfaces prompt the participants to give a different mean level of response (between 0 and 5), with a common variance. We will only apply the test to the overall' question, to give an indication of the significance of the results. Therefore there are three significance tests to be performed, all using the overall' question responses: Blackboard vs. Small, Blackboard vs. Wide, and Small vs Wide. The Student T' distribution is used to test whether the two measured mean response values are significantly different. Writing the Blackboard and Small mean responses as μ_b and μ_s, with variances σ_b^2 and σ_s^2, the test statistic t_{bs} will be defined as

$$t_{bs} = \frac{\mu_b - \mu_s}{\sigma_{bs}\sqrt{2/n}} \qquad (1)$$

Where

$$\sigma_{bs} = \sqrt{\frac{\sigma_b^2 + \sigma_s^2}{2}} \qquad (2)$$

and n is the number of participants, 11. Corresponding expressions can be derived for the other two significance tests, t_{bw} and t_{sw}. Thes t values are then used to obtain the p-values, which is the probability of obtaining this difference (or more), given the Null Hypothesis. These values can be either one-tailed (looking only one way for a difference) or two-tailed (looking both ways for a difference). In all cases the number of degrees of freedom (used in calculating the p-values) is $2n - 2 = 20$. These values are expressed as percentages in Table 1.

The last two columns indicate that the only highly significant comparison that could be made is between the responses about the Blackboard Interface and the responses about the Wide Interface: there is only a 0.05% chance that this degree of difference would have been generated by chance. The other comparisons are on the threshold of being significant: more participants would be needed to conclusively test the hypotheses that, for example, the Small Interface is preferred over the Blackboard, or that the Wide Interface is preferred over the Small one. Fig. 3 depicts the mean values of all responses, adding arrows to indicate one standard deviation in the sets of responses. It is interesting to note that the response to the Small Interface has smaller variance than the other two. This may be for example due to the fact that it was always the first component of the usability trial.

6 CONCLUSIONS AND FURTHER WORK

This paper has investigated how the layout materials and resources can have an impact on the effectiveness on a programming activity. An Interface has been developed, to allow Programming Tasks and Learning Material to be presented in a single browser. The lecturer's observation of the students' interaction had a big influence in the design, implementation and evaluation of the project. The feedback from the students has given useful guidelines for future improvements and evaluations. During the design of this learning experience every effort was made to increase students' motivation to understand and learn more about programming.

To ensure that this is a sustainable resource, the academic content needs to be easily edited and appended. There is a balance of arguments on this point. On one hand, a relatively simple HTML markup file (with style sheets and scripts stored elsewhere) provides a standalone resource, easily deployed and hand-edited. On the other hand, the avoidance of content duplication, the provision for authors unfamiliar with HTML, the requirements for concurrent editing and version control, all mitigate in favour of some central content management system that uses server-side processing to dynamically generate the page.

The current design includes two of the three required resources. If server-side processing is included, it would be possible to include other resources such as a program editor within the overall design. This has the advantages of further simplifying and controlling the layout of the learning resources, allowing student work to be centrally stored and monitored, and also providing a simple Interface with web programming environments.

The reference resources currently are Glossary and Common Tasks. Further resources could also be considered for inclusion such as third party reference texts, web-logs (blogs') for discussion by students about the current task, and other learning media such as audio descriptions and animations of the problem and the solution.

The development of two Interfaces gives the students a choice. Students with access to a wide screen can choose between the Wide and the Small Interfaces by simply resizing the browser. The Wide Interface will be appropriate for students who wish to see the Learning Material all the time. The Small Interface will be appropriate for students who wish to see the Learning Material when they need help. Also, students who wish or have to use smaller screens, the Small Interface would be more suitable.

A usability study was conducted using three different layouts of materials and resources to investigate the impact on the effectiveness on a programming activity. The use of the Interface to solve the tasks and find relevant information may depend on specific differences between students. However, all the participants managed to use the Interfaces and organise the IDE window as required and anticipated. The way the participants arranged their windows when attempting to solve the Task using the Blackboard Interface was also as predicted. The problems included too many open windows at one time, and difficulty finding help for the Blackboard Interface.

During the usability study, participants lacking in prior knowledge used the resources more than students who had a higher prior knowledge level. One of the main observations was the use of Learning Material as a help system: it is important to understand how the help is being used, as this is in itself a skill. The Interface should have a valuable help system, rich in content. Help-seeking behaviour may reflect students' attitudes about learning, their achievements and goals. Learners can be helped to be more productive and encouraged towards an independent way of thinking and problem-solving.

The students' preferences for the Interface differed in accordance with their level of experience, interest and expectations. The overall experience of using the Wide and Small Interfaces by the students was positive. The fundamental principal is that the layout of resources should be maintained to reflect their role in the student's learning activity: the editor, the instructions, and the background materials that assist them. It is hoped that this structure can provide a useful learning framework.

REFERENCES

Ardito, C., Costabile, M., Marsico, M., Lanzilotti, R., Levialdi, S., Roselli, T., and Rossano, V. (2006). An approach to usability evaluation of e-learning applications. *Universal Access in the Information Society*, 4(3):270–283.

Bennedsen, J. and Caspersen, M. (2005). Revealing the programming process. *ACM SIGCSE Bulletin*, 37(1):186–190.

Dabbagh, N. (2005). Pedagogical models for E-Learning: A theory-based design framework. *International Journal of Technology in Teaching and Learning*, 1(1):25–44.

DiMaggio, P. and Hargittai, E. (2001). From the Digital DividetoDigital Inequality: Studying internet use as penetration increases. *Princeton Center for Arts and Cultural Policy Studies, Working Paper*, 15.

Govindasamy, T. (2001). Successful implementation of e-Learning:: Pedagogical considerations. *The Internet and Higher Education*, 4(3-4):287–299.

Hodges, C. (2004). Designing to motivate: motivational techniques to incorporate in e-learning experiences. *The Journal of Interactive Online Learning*, 2(3):1–7.

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, pages 53–58.

Kölling, M. (2008). Using bluej to introduce programming. *Reflections on the Teaching of Programming*, pages 98–115.

Lee, M., Pradhan, S., and Dalgarno, B. (2008). The effectiveness of screencasts and cognitive tools as scaffolding for novice object-oriented programmers. *Journal of Information Technology Education*, 7:62–80.

Lif, M., Olsson, E., Gulliksen, J., and Sandblad, B. (2001). Workspaces enhance efficiency–theories, concepts and a case study. *Information Technology & People*, 14(3):261–272.

Lind, M. (1994). Effects of sequential and simultaneous presentations of information. *Report no. 9, CMD, Uppsala University*.

Mayer, R. and Moreno, R. (2003). Nine ways to reduce cognitive load in multimedia learning. *Educational psychologist*, 38(1):43–52.

Nielsen, J. (1994). Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems*, pages 413–414. ACM.

Techsmith, M. (2010). Usability Testing for Software and Websites. http://www.techsmith.com/morae.asp.

Wolf, C. (2003). iWeaver: towards 'learning style'-based e-learning in computer science education. In *Proceedings of the fifth Australasian conference on Computing education-Volume 20*, pages 273–279. Australian Computer Society, Inc.