

Asynchronous JavaScript Technology and XML (AJAX)

Chrisina Draganova

*Department of Computing, Communication Technology and Mathematics
London Metropolitan University
100 Minories, London EC3 1JY
c.draganova@londonmet.ac.uk*

Abstract

AJAX is a web development technique for building responsive web applications. The paper gives an overview of the AJAX technique and explores ideas for teaching this technique in modules related to Internet technologies and web development. Appropriate examples for use in lab sessions are also suggested.

1. Introduction

A new type of dynamic web applications with richer interactive and graphical capabilities has emerged recently. These web applications are developed using a technique called AJAX (short for Asynchronous JavaScript Technology and XML), which is based on well-established technologies. While in a traditional web application the user waits for a response and gets an entire page reloaded for each request, with an AJAX web application the user is not aware when the request is sent and he can continue to interact with the web browser. The communication with the web server happens in the background and the response returned to the user's browser contains only a small amount of data, which is used to update the page without reloading.

The constant change in the technologies used to develop web applications requires continuous revision of the web related modules in order to prevent them from becoming obsolete. Before including a new approach or technology in an existing module, several considerations should be taken in account, including the student's background, the cohesiveness with the other technologies already taught in the module, the maturity of the technology, the available resources and the technical support. In addition to these considerations ways of linking the technologies to relevant theoretical concepts in appropriate contexts should be explored to improve the overall student's understanding.

This paper aims to share some ideas of how to introduce the AJAX technique in modules related to web development. The paper is organised in five sections including the introduction. The second section gives an overview of the AJAX technology; the third section includes a practical example demonstrating the basic implementation issues, knowledge and skills required for learning the AJAX technique; the fourth section addresses the educational implications and the last section gives the conclusions.

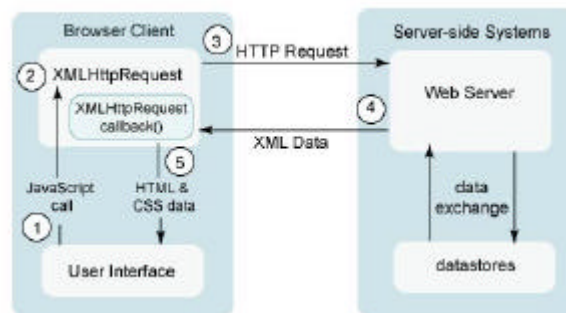
2. AJAX overview

AJAX (short for Asynchronous JavaScript Technology and XML) is a technique, which combines several well-established standards. It enables building more interactive web applications behaving in a similar fashion to traditional desktop applications. James Garret [1] coined the term AJAX in 2005, but the technique was known before as remote scripting with hidden frames or iFrames. Simple examples related to remote scripting without using the new AJAX technique can be found in [2], [3] and [4].

In a typical web application the user action such as submitting a form or clicking a hyperlink triggers an HTTP request, which is sent by the web browser for processing to the web server. The web server processes the request and it sends back a response, which contains an entire web page to be loaded on the user's web browser. While this happens the user is waiting for the response and his interaction with the

web site is blocked. There are many cases where reloading a whole page is not necessary such as real time data validation, auto completion, master details operations, sophisticated user interface controls, refreshing data on a page and server-side notifications [5]. AJAX provides a way of dealing with such cases, based on a model, which enables sending a request to the web server and receiving a response containing only a small amount of data in the background and refreshing only a specific part of the already loaded web page.

The main key to the AJAX technique is the so-called XMLHttpRequest object, which recently became part of the JavaScript technology [5] and all mainstream browsers support it. To build an AJAX solution three technologies are required: HTML/XHTML, DOM and JavaScript [2]. JavaScript language is used to develop the so-called AJAX engine or functionality that handles the communication with the web server, the user actions and the updates of the web page content. The DOM model of the XHTML pages enables dynamic updating of a loaded page and HTML/XHTML languages are used for content representation of the web pages. In addition to these technologies XML can be used as a standard format for data exchange and CSS for defining the style of the XHTML/HTML elements. Figure 1 illustrates how the AJAX interaction model works.



Source [6]
Figure 1

The following sequence of steps occurs in a general AJAX request [6]:

1. The user generates an event (e.g. entering input or clicking a button). This event triggers a call to a JavaScript function.
2. An XMLHttpRequest object is created and configured with a request parameter that includes the ID of the component that generated the event and any value that the user has entered.
3. The XMLHttpRequest object makes an asynchronous request to the web server, i.e. while the request is made the user interface is not blocked. The server processes the request, exchanges any data required from the data store.
4. Although Figure 1 shows that the web server returns XML Data in fact the server can return any fragment of data, which typically is an XML document or a plain text containing the result.
5. The XMLHttpRequest object calls a callback() function, it receives the data and processes the result.
6. The HTML DOM is updated.

Adopting the AJAX model can bring many benefits to the usability of the web applications by making the user interface more responsive, faster and graphically richer [7], [8]. However this approach has some disadvantages such as increased complexity of designing, developing and debugging [8], browser incompatibilities due to the different JavaScript implementations [1] and lack of suitable tools and frameworks for building this new kind of web applications [7].

The main factor for AJAX becoming so popular is the adoption of this technique by **Google** in their projects: **Gmail** - gmail.google.com, **Google Maps** - <http://maps.google.com/> and **Google Suggest** <http://www.google.com/webhp?complete=1&hl=en>. Other popular web sites using the AJAX technique include **Amazon A9** - <http://a9.com>, **Yahoo! News** - <http://news.yahoo.com/>, **Yahoo Flickr** - <http://www.flickr.com/> and **Orkut** - www.orkut.com. There are also a number of web sites, which use the AJAX technique to develop functionalities typical for desktop applications such as **Writely** - <http://www.writely.com/> - web word processor and **Kiko** - <http://www.kiko.com/> - online calendar. These applications have raised the expectations of the users from the web and pushed the companies and developers to adopt the new AJAX technique.

3. AJAX example

The best way to demonstrate the AJAX technique and to see the knowledge and skills required for its use is to look at a practical example. One simple and interesting Ajax application is Google Suggest, which implements a typical desktop functionality of suggesting values as the user types. The example given in this section simulates to some degree the behaviour of Google Suggest. It is a simplified version of the “Autosuggest Text Box” from Chapter 7 [2]. It implements the “Typeahead” [2] functionality, which helps the user to type the name of a county in England. More precisely described the functionality includes the following:

- ? As the user types in a text box the rest of the text box fills with the name of a county that starts with the letters that the user has typed already.
- ? As the user continues to type the text box automatically adjusts its suggestion.
- ? The suggested text appears selected (highlighted).

The example can be tested on <http://www3.unl.ac.uk:8188/draganov/autosuggest/autosuggest.htm> and its source code can be downloaded from [9].

The example consists of three components: an HTML component that gives the user interface, a JavaScript component that contains the AJAX engine and a server side component which handles the HTTP requests/responses and returns the relevant suggestions. In this example XML and CSS are not used in order to focus on understanding the principle steps in using the AJAX technique.

The HTML component contains a text box (see Figure 2) with an event attribute `onkeyup` set to a call of the JavaScript function `handleKeyUp(event, this.id)`. Two parameters are passed in this function: the event object and the value of the attribute `id`. As the user types the `onkeyup` event fires after a change is made to the text box. The `autocomplete` attribute of the text box must be turned off in order to eliminate the default autocomplete functionality provided by most of the browsers.

```
<p><input type="text" id="txtAuto" onKeyUp = "handleKeyUp(event, this.id)" AutoComplete = "off"/></p>
```

Figure 2: autosuggest.htm

The JavaScript component consists of three functions (see Figure 3, 4 and 5). For simplicity the code shown in Figures 3 and 4 is applicable only for the Mozilla web browser.

The first function `handleKeyUp(objEvent, txtID)` detects the character keys using the `keyCode` property of the event object and calls the second function `requestSuggestions(txtID)` passing the `id` of the text box as a parameter.

```

function handleKeyUp(objEvent, txtID) {
    var keyCode = objEvent.keyCode;
    // detects character keys
    if (!(keyCode < 32 || (keyCode >= 33 && keyCode <= 46) || (keyCode >= 112 && keyCode <= 123)))
        requestSuggestions(txtID);
}

```

Figure 3: handleKeyUp()

The requestSuggestions(txtID) function is the main element in the implementation of the AJAX technique. An XMLHttpRequest object is created and the method open() is called to initialise the object. The open() method takes three arguments: Request Type – GET or POST, URL – a string indicating the URL to send the request to and a Boolean value indicating whether the request would be asynchronous. When this Boolean value is set to true the request is sent asynchronously enabling the execution of the JavaScript code to continue without waiting for a response and the user to continue to interact with the browser. Next an event handler onreadystatechange has to be defined. This event handler is set to anonymous function, which checks the readyState property of the XMLHttpRequest object. There are five possible values of this property, which are described for example in [2]. Every time the readyState property changes its value, the onreadystatechange event fires and the respective event handler is called. Value 4 indicates that the request has been completed and all data from the response has been received and the connection has been closed. When this happens the function typeAhead(suggestion, txtID) is called passing the text returned in the response and the id of the text box to be updated when the returned suggestion is not empty. Finally the request is sent by calling the send() method. This method requires a string parameter for the request body. The GET request does not have a body, therefore the parameter passed is null.

```

function requestSuggestions(txtID){
    var objEl = document.getElementById(txtID);
    var httpRequest = null;
    if (objEl.value.length > 0){
        if (typeof XMLHttpRequest != "undefined")
            httpRequest = new XMLHttpRequest();
        else
            alert("No XMLHttpRequest object available. This functionality will not work.");
        //build the URL
        var sURL = "http://www3.unl.ac.uk:8188/draganov/autosuggest/suggestion.jsp?txtInput=" +
            encodeURIComponent(objEl.value);
        //open connection to states.txt file
        httpRequest.open("get", sURL , true);
        httpRequest.onreadystatechange =
            function () {
                if (httpRequest.readyState == 4) {
                    var suggestion = eval(httpRequest.responseText);
                    if (suggestion.length > 0)
                        typeAhead(suggestion,txtID);
                }
            };
        httpRequest.send(null);
    }
}

```

Figure 4: requestSuggestions()

The third function typeAhead(suggestion,txtID) changes the value of the text box to the suggested value and selects that part of the text that was not typed by the user.

```

function typeAhead(suggestion,txtID){
    var objEl = document.getElementById(txtID);
    var txtValue = objEl.value;
    objEl.value = suggestion;
    if (objEl.setSelectionRange) //Mozilla select
        objEl.setSelectionRange(txtValue.length, objEl.value.length);
    objEl.focus();
}

```

Figure 5: typeAhead()

The last component is the server side JSP script. It handles the HTTP request and returns the relevant suggestion. To simplify the example the names of the counties are hard coded in an array. In real applications there will be a database tier that stores the names of the counties. The content type is set to `text/plain` since only text is sent back. This component could be implemented in any server side scripting language.

```

<%@ page contentType="text/plain; charset=UTF-8" %>
<%
    String txtUser = request.getParameter("txtInput");
    String[]counties = new String[] { "Avon","Bedfordshire","Berkshire","Buckinghamshire","Cambridgeshire","Cheshire","Cleveland",
        "Cornwall","Cumbria","Derbyshire","Devon","Dorset","Durham",..... };
    String suggestion = "";
    int i = 0;
    if (txtUser.length() > 0 ){
        while (i < counties.length && !counties [i].startsWith(txtUser) )
            i++;
        if (i < counties.length)
            suggestion = counties[i];
    }
    out.print("\n" + suggestion + "\n");
%>

```

Figure 6: suggestion.jsp

The implementation of the above example gives an indication of the knowledge, understanding and skills that are required to apply the AJAX technique. The first component requires basic knowledge of HTML. The second component requires good knowledge and understanding of the JavaScript language, event model, HTML DOM, the object-oriented paradigm and HTTP request/response model. It also assumes good programming skills. The last component requires skills in using a server side language and understanding of the development of dynamic web applications.

4. AJAX educational implications

In all of the computing courses offered at London Metropolitan University students are given the option to study modules related to web development and Internet technologies at each level: certificate, intermediate and honours. Each Internet related module builds on the knowledge and skills gained in modules from the previous levels including Internet, programming and database modules.

There are a number of reasons that one can list in support of the idea to incorporate AJAX topics in the Internet related computing modules. AJAX is a popular technique used with open standards and supported by all major browsers. It is focused on enhancing the user experience and consequently its use has expanded rapidly. Since the AJAX approach uses technologies and models that are already covered in the Internet, programming and database modules the students have the necessary prerequisite knowledge and skills to study the AJAX technique. Developing simple AJAX examples does not require any extra technical support and resources. Our previous experience from introducing topics such as Web Services, XML

and mobile web application development shows that students find it interesting and motivating to learn innovative and popular technologies. The benefits from introducing the AJAX approach for the students include: developing broader skills and problem solving techniques in web development, improving understanding of fundamental concepts related to event driven development, data tree structures, DOM, object oriented programming and acquiring knowledge in current trends in web development. Some challenges also could be expected such as overcoming the different level of understanding of the prerequisite concepts by the students, developing more suitable examples and finding an appropriate tool or a framework for rapid development of AJAX applications.

In the certificate level all computing students study two core modules in object-oriented programming with Java. There are two sessions in the second programming module [10], which considers the event-driven paradigm in the context of building GUI applications. In the Internet modules students gain understanding of key principles related to web applications and skills in design and development of static web sites. They are also introduced to the client side scripting language JavaScript, CSS and the XML technology. At this stage in order to prepare the students for using the AJAX technique examples that focus on using the HTML DOM, JavaScript Objects and the event model need to be considered. For instance, the W3 Schools [11] tutorials offer a large selection of appropriate examples.

The intermediate Internet modules cover development of data driven dynamic web sites. On completion of these modules the students are expected to be able to design and develop a multi-page web site allowing interaction with a database. The students gain experience of using a server side language and the HTTP request/response model. The AJAX technique can be introduced at this level by comparing the traditional method for developing dynamic web sites with the AJAX approach. Ideally one lecture followed by a tutorial and a workshop would be sufficient to cover the basis of this topic. Simple AJAX examples demonstrating real time data validation and periodic refresh for a mini chat room application based on the Ajax patterns suggested in Chapter 3 [2], have been developed. Exercises based on these examples that require small modifications are appropriate for the lab sessions to further develop the student's understanding of the AJAX approach.

The examples can be tested on <http://www3.unl.ac.uk:8188/draganov/realtimedatavalidation/inputdata.jsp> and <http://www3.unl.ac.uk:8188/draganov/chatroom/chatroom.jsp>, respectively, and their source code can be downloaded from [9]. These examples focus on implementing the basic AJAX engine in order to provide students with understanding of the fundamental idea of the AJAX technique.

In the honours level the module related to the Internet is focused on developing an understanding of enterprise level Internet software application programming issues. It covers in depth a number of topics including architectural design patterns, client-side and server-side programming, database connectivity, XML technologies and Web Services. For the first time the AJAX technique was considered last year in this module [12] in one lecture session. However, introducing the AJAX approach at intermediate level would enable the consideration of more advanced topics at honours level that are related to design patterns, web architectures and frameworks. This approach would be in line with the aim of the module to develop understanding of enterprise web application development. To demonstrate a real application of the AJAX technique students can be given an example of incorporating Google Maps in their web applications using the Google Maps JavaScript API [13]. In addition, more examples that demonstrate the best practices of using AJAX in web applications can be found in The Java BluePrints AJAX Components [14]. At honours level investigation of different tools and frameworks available for AJAX application's development is appropriate [15]. Adopting a tool for rapid development of AJAX functionality would enable the implementation of more complex examples, enhance the student experience and emphasise more the design principles rather than the implementation issues.

5. Conclusions

This paper has presented an overview of the AJAX technique for developing interactive web applications. Ideas of how to introduce the AJAX approach in teaching web development and examples for lab sessions have been suggested.

Incorporating popular technologies into the computing courses appears to motivate the students and in the same time it helps them understand better fundamental theoretical concepts. The AJAX technology requires relatively little new knowledge and can be easily included in teaching web development by focusing on specific parts of existing technologies and demonstrating appropriate applications. The challenge is the constant need of updating the material and selecting relevant examples and tools. However, the experience of learning new technologies and related applications has been rewarding and exciting. The material presented here could be used as an example of embedding emerging technologies in the computing curriculum through existing modules.

6. References

- [1] Garret J. J., Ajax: A New Approach to Web Applications, February 2005, <http://adaptivepath.com/publications/essays/archives/000385.php>, last visited 27.09.2006
- [2] N. C. Zakas, J. McPeak, J. Fawcett, Professional Ajax, Wiley Publishing, Inc, Prentice Hall, 2006
- [3] Brattli, T., Dynamic data using DOM and Remote Scripting, 2002,
- [4] Pallet D., Ajax & PHP without using XmlHttpRequest Object, <http://www.phpit.net/article/ajax-php-without-xmlhttprequest/>, last visited 27.09.2006
- [5] Murray, G., Asynchronous JavaScript Technology and XML (AJAX) With Java 2 Platform, Enterprise Edition, 2005, <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/>, last visited 27.09.2006
- [6] Murray, G., Ball, J., Including AJAX Functionality in a Custom JavaServer Faces Component, <http://java.sun.com/javase/javaserverfaces/ajax/tutorial.jsp>, last visited 27.09.2006
- [7] Paulson, L.D., Building rich applications with AJAX, Computer, p 14 - 17, Vol 38, Issue 10, Oct 2005
- [8] Smith, K., Simplifying Ajax-Style Web Development, Computer, p. 98-101, Vol 39, Issue 5, May 2006
- [9] http://www3.unl.ac.uk:8188/draganov/ajax_examples/ajax_examples.zip
- [10] Fisher, K, Graphical User Interfaces, <http://languages.londonmet.ac.uk/java2/week5/index.htm>, last visited 26.12.2006
- [11] W3 Schools, www.w3schools.com, 2006
- [12] Campbell, M., Lecture slides, <http://www.city.londonmet.ac.uk/~cambridg/qc310/AJAX/qc310-wk10B-Ajax.ppt>, last visited 27.09.2006
- [13] Google Maps API Version 2 Documentation, <http://www.google.com/apis/maps/documentation/>, last visited 26.12.2006
- [14] Java BluePrints AJAX Components, <https://blueprints.dev.java.net/ajaxcomponents.html>, last visited 27.09.2006
- [15] Ajax Frameworks, http://ajaxpatterns.org/Ajax_Frameworks, last visited 27.12.2006

Can the Fine Arts Inform Software Development ?

(From 20th Century Russian Constructivism to 21st Century Java)

Colin B. Price

Computing Section, Business School, University of Worcester,
Worcester, WR2 6AJ, UK. c.price@worc.ac.uk

Abstract Can a study of Fine Art, paintings and Art Education help us to improve our teaching of software development in Java? In a recent study at the University of Worcester, we have found the answer is “yes”. Through consideration of early 20th Century abstract art and through studies of the processes of Art education, and through development of programming tasks, we have concluded Art and Design can motivate our students to appreciate the principles of OOP and to learn to write code in Java. This paper presents the rationale for our approach, gives details of the implementation and briefly reports on the effectiveness of our approach. This work has been conducted during delivery of an “Advanced OOP (Java)” programming module at final-year Undergraduate level, and during a Masters’ “Conversion” OOP (Java) module. It is a pilot study, conducted during a single semester. Case studies, including learning materials and student work are provided at associated web-pages.

1. INTRODUCTION

Software development is essentially a *creative* process, involving the construction of an artefact which does not yet exist. This is not engineering, which is essentially the production of “variations on a theme” (a new bridge or concrete structure). The engineer is often concerned with the application of new materials. The architect possesses qualities shared by the artist, but of course grounded in the need for functionality. So too is the software engineer; like the architect, he employs the current materials (language) and reflects upon the prevalent styles (patterns), then designs and constructs his structures (programs).

Parallels between Artists and Programmers may be further explored: How do we educate our Artists, and programmers? A crucial part of Art education is the study of great Masters, through a study of their paintings, and through an appreciation of their lives and social context. The “programming” parallel is easy to see: The study of Masters corresponds to “reading code” (of the “greats”, see Section 7), appreciation of lives and contexts corresponds to a study of the historical development of the Babel of computer languages (e.g., from *Smalltalk* to *Java*). In Art education, substantial time is given to critical analysis and peer review. This corresponds to team programming and debugging activities. While the work of a “software Master” is like a Bach fugue, a novice programmer must also engage with this music; any meaningful computer program is not a collection of isolated chunks of code, but many and highly interrelated chunks. This is even true of OOP. See Section 7 below.

This paper will strive to show that there are synergies between Art and computing, and that these synergies are *constructive*, and that we should acknowledge an “arrow of information” from Art and Design into Software Development. Richard Gabriel, Distinguished Engineer at Sun Microsystems, is also an advocate of Art informing software development; he has proposed the development of a “Masters of Fine Arts in Software” [2]. In our project we introduce our students to the principles of OOP via a study of early 20th century abstract painters, especially the Russian Constructivists and Suprematists. The materials were used with two classes: The first, a final year BSc. Computing class, had prior exposure to OOP. The second, an MSc. Computing (conversion) class was meeting OOP and Java for the first time. We constantly compared the learning in these two classes which used the same resources.

We present web-pages containing both learning materials and student work at the url [14] which explore the ideas presented in this paper.

3. THE NATURE OF PROGRAMMING

Why do we create programs? What are programs? How should we best create programs? These are in effect *metaphysical* questions. Reality returns when we embrace our “subject benchmark statement” [15] which highlights programming as a core element of our computing curriculum. So does Denning in his report on the “discipline” of computing [1]. But what about OOP; is this a language or a paradigm? We view OOP as primarily a methodology of thinking and not a paradigm of programming; we emphasise the cognitive aspect, *c.f.* [19]. This methodology *induces* from the special to the general. In our context, a discussion of a number of Artist’s paintings leads to the classification of objects within that painting. When these classes are established, instances may be created in code, and placed on the Canvas; this is a process of *deduction*. These processes of *induction* and *deduction*, used in ordinary life in thinking and reasoning provide a powerful methodology for programming. In other words, learning OOP should be driven by this methodology and not by the Java (or other) language?

Our learning materials are essentially graphical; the “console” is used for student-elicited debugging messages. The rationale for using an interactive graphical approach to learning and teaching programming is not difficult to support: A graphical visualisation of the students’ modification of code is rapid and information-rich. But also, in this context of an inductive-deductive relationship with Art, students are motivated to be creative. The question remains how to implement an interactive graphical learning environment? Some educators support the use of specially crafted graphical libraries to hide away the “complexities” of Java AWT and Swing, such as the “Nice Graphics Package” developed at Brown University. We choose not to do this, but to present the AWT and Swing packages as a learning experience for our students, involving a “drilling-down” into the Java API class tree, discovering where attributes and methods reside and how they are inherited. Activities based upon a study of the Java classes are themselves metaphors for the study of the Artworks of the Grand Masters.

4. ALGORITHMIC ART – WHAT OUR PROJECT IS NOT

This is certainly not the first project to argue a connection between Art and programming. But previous work has emphasised the use of programs to generate Art, e.g., the *Processing* project at MIT [11] and the inclusion of a Java programming module in a Fine Art course [15]. We refer to these approaches, which in effect produce visualizations of algorithms as belonging to the “Algorithmic school”. Algorithmic art is like an arrow flying from programming to art, producing (viable) art by programming (algorithms), such as classic visualizations of Mandelbrot and Julia sets. Recent more subtle attempts [9,11] have not really shifted the paradigm. Our work effectively proposes Art as a *metaphor* for OOP: Looking at an Artwork involves categorization, discovering base classes and dependencies, inheritance. Metaphor and categorization have been studied by the linguist Lakoff, [7], a study of his theories may be useful to understand the learning and teaching of OOP. Similarly the “Cognitive Dimensions framework” of Green [3], which is designed to evaluate the usability of information artefacts, could be used to identify how intended learning outcomes are actually supported by the programming tasks. A good analysis of the methodologies associated with the construction of Algorithmic Art is provided by the artist Roman Verostko [20].

5. OVERVIEW OF THE MODULES

The 12 weeks of module delivery are split into two phases. Phase 1, comprising 5 sessions, established the principles of OO Programming. Despite the concerns of some educators, we definitely take “Objects first”. Here the students worked through 5 activities comprising a several tasks. Tasks contained base code, supplied by the tutor, which the students modified and extended. This provides a rich environment for investigation. All students started off with the same base code, allowing sharing of understanding and peer-review of individual’s development. In Phase 2 the more technical aspects of Java were discussed; from files, images, to RMI and the 3D JOGL library [4]. Also, techniques used in Phase1 (such as Swing events) were discussed. This approach to learning is iterative; students have a rich space to be constructive and the tutor has opportunity to reinforce understanding of language specifics.

A discussion of our pedagogy has been given elsewhere [12]. The fundamental idea is to provide a *rich* learning environment which may support many different learning styles, especially *active learning* where each student constructs their own knowledge and understanding. This process is *iterative*, as noted by Soloway, we should provide an environment which (i) proposes several solutions to a problem, (ii) provides several alternative decompositions of a problem through exploration and experimentation, (iii) produces a synthesis of a solution through analysis of these alternatives, (iv) allows implementation of the solution in software, and (v) encourages reflection on this process [18,19].

To model the actual iterative dynamic process of learning, we were influenced by a number of approaches. The first was Kolb's "learning circle" [6] which models learning as an interactive loop from (i) experimentation (ii) to obtain a concrete experience, (iii) and reflection on that experience and finally (iv) to abstraction. We implemented this by requiring the students to maintain a learning journal, where at the end of each "activity" they were invited to reflect upon their work during the activity. Kolb's abstraction stage was obtained via class discussion. The second influence is the relationship between processes of action and reflection proposed by Schon [16].

5. PHASE-1 ACTIVITIES

In the first session of instruction, we looked at the work of Malevich, (e.g., Fig.1). A class discussion of several of his works soon led to the identification of similar, common elements. A classification of these elements was made, and so we converged to the notion of "Class" (elements which shared common "attributes" (such as form) and then the notion of "objects" as "instances" of those classes actually painted on the canvas. Suddenly, the need for a "canvas" class emerged! We also discussed the visitors to an Art gallery and their relationship to the Artwork. From this emerged the need for a "GUI" class which captures the interaction of the viewer with the canvas. Our final classification became (i) Art components (or "objects") drawn on the (ii) "canvas", which itself was viewed by (iii) the "visitor" via a GUI.



Figure 1. Malevich Self Portrait

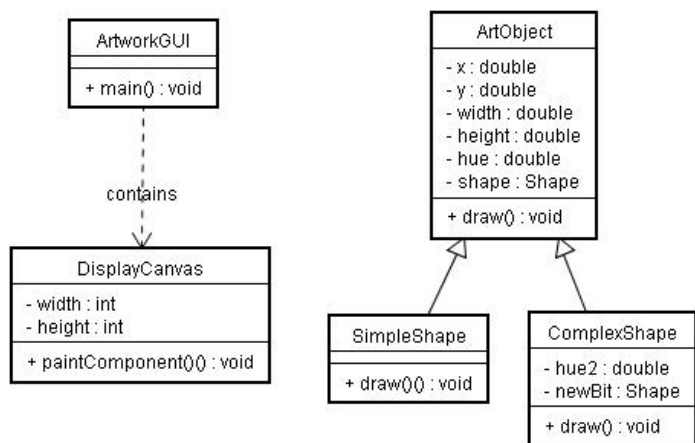


Figure 2. Classes and Dependencies

In terms of OOP, this generated three "base" classes: *ArtObject*, *DisplayCanvas*, and *ArtworkGUI*. The discussion moved on to identify the "intrinsic qualities" of these classes (e.g., *width* and *height* of the *DisplayCanvas*) and so to the concept of attributes. Then a discussion of methods debated where the "draw()" action should be located. Conclusion - within the *ArtObject* class; each element is responsible for drawing itself. Revisiting the classification of *ArtObjects* soon led to the

notion of a hierarchy, i.e., sub-classes and inheritance. Following these discussions, a small set of Java classes was coded by the tutor which were used to generate a series of activities and tasks where the students explored the principles of OOP discussed. As mentioned above, programming is a Bach fugue, a coordinated interrelation of melodies. The initial tasks asked the students to instantiate several *ArtObjects*, explicitly in the *DisplayCanvas*, (later this would be done using arrays or Collections). Using the constructors, they explored the meaning of *parameters*, setting different locations, sizes and colors, whether the object was filled or outlined. Then they were invited to consider how to extend a *SimpleShape* into a *ComplexShape* discovering the meaning of “*extends*”, inheritance and overloading (via the *draw()* method).

At this point we introduced the array data structure, as a means of collecting the multiple objects were instantiation on the canvas into a coherent whole. A 1D array produces a line of *ArtObjects* (see Fig.3(a)). Of course the loop program construct needs to be introduced here. Then the selection “if { } else{ }” construct was introduced, to vary the color or the *ArtObjects* produced, or the size of the *IAOs*. The natural progression to a 2D array followed, with students experimenting with mathematical functions, conditionals to produce pleasing 2D canvases (Fig.3(b)). Finally, we discussed the array, its limitations (fixed type, and fixed size) and the need for the *ArrayList* was proposed. Further tasks explored the use of the *ArrayList* to support the instantiation and drawing of *ArtObjects*.

The introduction of the array before the *ArrayList* may be questionable. Some may suggest that the *ArrayList* is more “OOP” than the array. Maybe, but we have found that students are able to conceptualize a bounded space of “Excel-cells” quite easily, and that this understanding transfers directly into our graphic representation of a 2D array of *ArtObjects* (or *JButtons*, etc.).

Further activities within “Phase 1” introduce (i) the construction of GUIs using Swing, (ii) dynamic and interacting *IAOs*, using *Threads*, (iii) *Applets* to publish cool constructions on the Web. One session is devoted to the study of the game “Pingpong”, since despite the lack of emphasis in this module on game programming, many students request support for this goal.

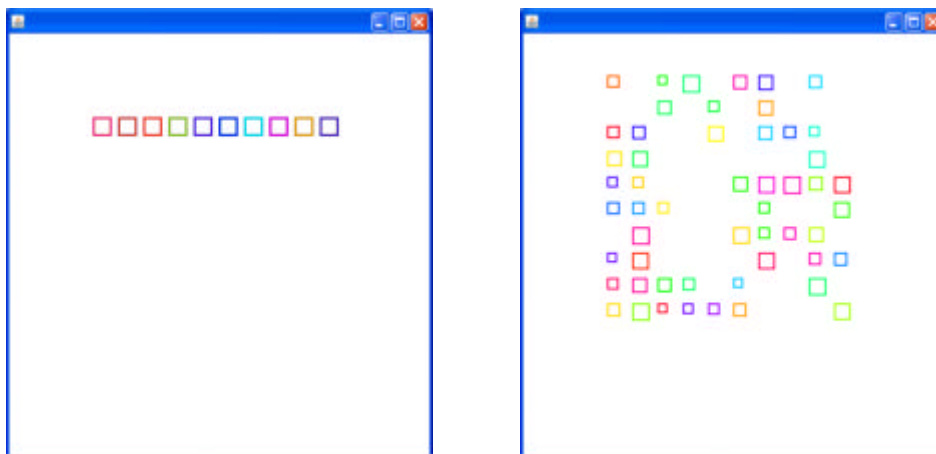


Figure 3. (a) Left shows the depiction of a 1D array with random association of hue, (b) Right shows depiction of 2D array with selection structure driven by random numbers

6. PHASE-2 ACTIVITIES

There are two intentions in Phase 2. First to review those technicalities of Java used in Phase 1 (such as Swing events, loading of Files or Images, etc.) and to discuss them in detail. Second is to introduce new Java technology, such as RMI or JMF, both of which form a substrate for our research and development at Worcester. Phase 2 is very much directed by the students’ needs in developing their final projects, yet on the other hand, exposure to new material may be of relevance. This material is presented “formally”, by short tutor input supported by on-line activities which students may chose to explore if they feel it is appropriate to their project. The pedagogy is simple; to provide a diversity of materials and approaches to support individual learners, but also, in discussion, to identify those

technical areas which may be of common use and so need to be discussed in class, i.e., a combination of autonomous, peer-interactive and class-based learning.

7. READING CODE

Here students were taken into a computer-less learning space and presented with A3 sheets containing computer code on paper. A tutor-led discussion of OOP was held, with the class reading, analyzing and discussing the code. Here we invoke the metaphor of “studying the Masters”. Both Masters of Art (Cezanne, Picasso, Malevich, Kandinsky and Klee), and Masters of Software (Wirth and Klein?). The focus shifts from experimentation and visualisation, to analysis, understanding and verbalisation. Interestingly, verbalisation emerged as a significant issue: Several students who understood the concepts (as demonstrated by their ability to *use* them) had difficulty in finding the right words to express their understanding. This is an issue which we must follow up. Both BSc. and MSc. students were given the possibility of using UML Class and Sequence diagrams to aid in this discussion (using *Jude*, *BlueJ*, or *JGrasp*), but, interestingly, no single student engaged with this; they found the “reading code” activity itself sufficient and of significant education value. These “reading code” activities seemed to concretize abstract concepts in a complex situation. Students showed that they understood principles of OOP and of Java syntax when these were incrementally introduced, but yet had difficulty in assimilating these concepts (i.e. to write code) for a complex application of their choice. We return to the metaphor of the Bach fugue; so much is going on at once! Through reading code they were able to investigate and reinforce their understanding of many concepts; inheritance, polymorphism etc. The hardest concept seemed to be the sequence of message-passing (method calls) between a plethora of objects (and the referencing required to enable these calls). Here, reading code really helped. This activity was also transferred to an “assessment item” for the students’ final portfolio, much appreciated, and motivating!

The class discussions based on reading code were useful in revealing the grasp of students’ understanding. These discussions were lively and fundamentally peer-to-peer, with the tutor often relegated to the position of a “linesman”, rather than a “referee” or “manager”. A lot of learning was going on in these activities, indeed students requested additional “reading code” sessions. However, we suggest that “reading code” activities cannot exist in a vacuum but should be contextualized, in response to both the tutor-provided activities, and the student-experienced difficulties.

8. A CONFLUENCE OF UNDERSTANDING

Space does not allow us to present a complete mapping between Art concepts and programming concepts, structures and syntax in this paper. You are invited to visit our supporting web-pages at [14] which provide “case studies” of how concepts of Art may inform our teaching of OOP, as well as student work. As artists, we paint, as programmers, we code. There are here two different media of expression, but one shared goal, to produce Artworks which are viable. Consider the Genetic Algorithm (GA) paradigm, a well known technique in Artificial Intelligence. Applied in our context, GAs step outside the action of “painting” and raise the question “how to paint”. GAs produce a “family” of Artworks, each unique but sharing familiar aspects, *concepts* of Art. What are these “concepts” which are embodied in the GA and other code? These are defined by the established artist, embodying years of experimentation and discovery. Their realization as *software Art* is nothing more than a realization of the artists’ own *concepts*. For example, Klee has taught us to pay attention between the *controlled* and the *uncontrolled* [5]; a dialectic between the specification of the geometrical and the provisional. In one of our software paintings, we locate a juxtaposition between programmed composition and user interaction: Rectangular *fields* define localized dynamics of “Interactive Art Objects” (IAOs); the artist inserts IAOs onto the canvas which interact with these fields and a dynamic *emerges*. The computer “controlled” is situated against the artist’s “uncontrolled”, (which derives from feeling or intuition). The Artist’s expression of free aesthetic conception is realized through brush strokes and color mix on a physical canvas. This is constrained by the physical reality (the physical texture of the brush, the mixing of colors and generation of optical

percept). But also by the physics of the real world. Consider the work of Jackson Pollock where gravity colludes with his aesthetics to create an Artwork. Can we programme this? Can we parameterise this? Of course! This is a simple interactive simulation task.

Our employment of *fields* of activity on the canvas was motivated by Piet Mondriaan's concept of "dynamic equilibrium", based on a dialectic between opposites and equilibrium. Mondriaan, Klee, Malevich and Kandinsky have enthused us to produce many "ArtApplets" to explore the concept of visual dialectic. The arrow from Art to software is defined by the processes of creating Art. It may start with a discussion of how to draw lines, how to choose colors, or about which forms are fundamental. But, fundamentally, an understanding of aesthetics, relationships, and style must precede commitment to computer code. In a certain sense, this involves a translation of the "procedures" of an Art concept into a meta-level of code, as described via OOP.

9. EVALUATION AND CONCLUSIONS

Space does not allow a detailed presentation of the results of our evaluations. Here we describe only a shadow. Our BSc. students had already been exposed to OOP and Java, so we used a pre and post module Likert attitude test, to determine the effectiveness of the learning. All aspects of OOP were tested via questions, such as "How well do you understand *encapsulation*". The result of this study was that students had difficulty in knowing when to use a sub-class, or a parameterised the parent class. The MSc. class was evaluated via a short paper where students discussed the usefulness of OOP concepts. These papers were ranked by the tutor, and a set of criteria developed for assessing understanding of particular OOP concepts and. Re-reading the papers and applying these criteria led to assignment of marks reflecting understanding of the OOP concepts. Here one clear issue emerged, concerning encapsulation. Students suggested that the encapsulation of attributes seemed draconian; access to an attribute (when made "public") often made more sense than the need for a "get()" method when that attribute was made private.

It is also interesting to consider anecdotal evaluations of our approach. Following an end-of-module evaluation, (conducted via a paper-based questionnaire), the following criticisms and praises were consistently noted by students. Criticisms related to specific OOP or Java concepts. These emerged during the teaching of the module, and we attempted to provide learning materials to help. Nevertheless, students flagged these as areas of concern, during the formal module evaluation: (i) Many students had difficulty in understanding the *ArrayList* collection, in particular the *iterator*. In response to this, we produced an ArtApplet demonstrating the construction, reading and modification of an *ArrayList*. This visual aid helped significantly (ii) While "message-passing", through method calls, apparently was understood, a complex sequence of calls was not appreciated, and students could not easily synthesise their own sequence of calls. This centered around the need for passing a reference to calling and called objects. We attempted to address this through the use of UML sequence diagrams, but students did not engage. However, "reading code" activities significantly helped here, where the sequence of messaging was discussed via print-outs of Java source. (iii) Students were unanimous in their fear of Java error and warning messages, and also of the API which required a significant investment of time, despite tutor support.

Points of praise fortunately outweighed these criticisms: (i) Students unanimously reported that their understanding of arrays and loops, (especially nested loops) was facilitated by the visual approach. (ii) The use of parameters to define the attributes of objects (often in conjunction with *Math.random()*) was appreciated. Given a task of producing an "appealing" Artwork, students experimented (and produced good code) using loops, conditionals and parameters. Despite our "objects-first" approach, this structural material was presented early on and was deemed to be very useful. (iii) The concept of dynamics as implemented through *Threads* was appreciated, students reporting that the immediate visualization of moving objects was beneficial to their understanding, e.g., of the *sleep()* method and of message passing, to "step()" and to "draw()" a list of objects.

When specifically asked to comment on the use of Art as a substrate for learning OOP and Java, students' comments were interesting. (i) Several had delved into the details of Java2D to produce

interesting dynamical graphical works. (ii) Others reported that analysis of abstract paintings, and synthesis of their own OOP-Java realizations provided an interesting and worthwhile experience. One student reported a real appreciation of the connection between programming and Fine Art, and a desire to take this further into research.

Students also reported that learning programming was not easy. They appreciated the efforts of their tutors, but suggested that learning programming is *complex*, so much must be learned at once; concepts, syntax, development tools (the IDE), modelling tools and engagement with the Java API documentation! Just like a Bach fugue!

The tutor's assessment of this module may also be of interest. We set up the student assignment as rather open-ended. There was a free choice of one "research" and one "programming" assessed component. Please drill into the module pages to review this [13]. Interestingly, all students, both BSc. and MSc. classes opted to produce a *computer game*. This is significant, since a previous delivery of this module took computer game development as its context. We had introduced computer games ("pong") into this delivery as a vehicle for discussion of threads, interactions and message-passing. This has clearly distracted our students from appreciating dynamic artworks as a *bona-fide* ground for learning OOP and Java. We were also disappointed by the negative response of our MSc. students in researching UML as a useful design tool, and also making a comparison of IDE's such as JCreator, JGrasp and BlueJ. They seemed to have become immersed in the technical details of creating working code in Java. Indeed, one student suggested that this module be split into two, a "theoretical" OOAD module and a Java programming module. Unfortunately, at the moment, we cannot resource this.

Concerning the autonomous learning of our students, we report positive results. Some students engaged with the Art context and immersed themselves in the study of 20th-Century art. One MSc. student has opted to research this approach as a topic of his dissertation. Another student investigated collision detection algorithms, to produce a realistic visualisation of fluid flow. Two students became engaged in the details of Java2D, one chose to investigate imaging, (transformations, rendering), the other the use of primitive elements to produce an Applet to investigate mathematical functions!

In conclusion, this has been a pilot study, an attempt to cross two disciplines, to investigate the tutor's interest in this trans-disciplinary area. The results have been clear: Each student has been able to identify a particular area of interest, where the generic concepts of artistic visualization have been realized, albeit in the production of graphical games. The cohorts involved were small; we expect around 15 students on the BSc. module and 6 or 7 on the MSc. While these numbers are conducive to research, we believe the results are generalizable to larger cohorts, when appropriate scaffolding is developed. All students produced graphic and dynamic pieces of software of high quality, and asserted their individual preferences. This is laudable. Yet no single student of programming has become an artist!

11. REFERENCES

- [1] Denning, P.J. (2000) Computer Science : The Discipline in Encyclopaedia of Computer Science, Ralston, A., and Hemmendinger, D (eds.) George Mason University, Fairfax VA.
- [2] <http://www.dreamsongs.com> (accessed Oct 20 2006)
- [3] Green, T.R.G. & Petre, M (1996) Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Jour. Visual Languages and Computing*, 7.
- [4] <https://jogl.dev.java.net/>
- [5] Klee, P., Spiller, J (ed.) (1992) Paul Klee Notebooks Vol 1 : The Thinking Eye William Stout, San Francisco CA.
- [6] Kolb, D.A., (1983) Experiential Learning. Experiences as the source of Learning and Development. Prentice-Hall.
- [7] Lakoff, G. (1993) The Contemporary Theory of Metaphor. In Ortony, A. (ed.) Metaphor and Thought. NY. Cambridge University Press.
- [8] Lewis, J., (2000) Myths about Object -Orientation and its Pedagogy. *ACM SIGCSE Bulletin*, Vol 32 No 1, March 2000
- [9] Maeda, J., (2004) Creative Code. Thames & Hudson, London.

- [10] Mitchell, W. (2001) A Paradigm Shift to OOP Has Occurred Implementation To Follow, *Journal of Computing in Small Colleges*, Vol.6 No.2 May 2001
- [11] <http://processing.org/>
- [12] Price, C.B., (2006) Learning Java, Learning Game Programming. Proc. JICC10, London Metropolitan University, Jan 2006.
- [13] Price, C.B. (2006) Module WebPages available at
- [14] Price., C.B (2007) JICC Supporting Web-Pages available at http://www.worc.ac.uk/departs/bm_it/colin/Resources/JICC11
- [15] <http://www.qaa.ac.uk/academicinfrastructure/benchmark/computing.asp> (accessed Oct 20 2006)
- [16] Schon, D.A., (1983) *The Reflective Practitioner*. NY. Basic Books.
- [17] SFMA Boston
- [18] Soloway, E. (1986) "Learning to Program Learning to Construct Mechanisms and Explanations," *Communications of ACM*, Vol. 29, No. 9.
- [19] Soloway, E., Spohrer, J., Littman, D., (1988) E unum pluribus: Generating alternative designs", in *Teaching and Learning Computer Programming: Multiple Research Perspectives*, R. E. Mayer (ed.), Lawrence Erlbaum Associates.
- [20] Verostko, M. (1988) Epigenetic Painting. Software As Genotype, A New Dimension of Art. *1st Symposium on Electronic Art (FISEA) Utrecht* Published in *Leonardo* 23:1 (1990)
- [21] Zhu, H., Yang, G., and Liu, Z., (1998) *Object Oriented Principle and its Application*. Publishing House of Changsha Institute of Technology.

Immersive Learning: Introduction to Programming – Java

Paul Sant¹ and Des Stephens²
Department of Computing and Information Systems
University of Bedfordshire

Abstract

First year programming has been consistently below the Faculty and University averages in SPOM (Student Perception Of Module) reports. An immersive learning style of delivery has been adopted in an attempt to improve student motivation and outcomes. This paper reports preliminary positive results.

1. Introduction

The Computing and Information Systems Department of the University of Bedfordshire has used Java as the first programming language since the academic year 2000 – 2001. This was initially taught as an introductory, first semester, 15 credit, programming language module for all students followed by a second semester, 15 credit, OOP module for all Computer Science and associated pathway students. The approach used is an objects later method utilising EditPlus2 as a simple IDE together with GUI I/O.

In the academic year 2004 – 5 this changed to a year long 30 credit programming module for Computer Science and associated pathway students with a parallel Visual Basic module for Information Systems and Computer Networking students. The rationale behind this change was to reduce the assessment load associated with modular schemes, provide a more integrated introduction to programming in Java for students for whom programming would continue to be a central part of their degree programmes, and also to provide for Information Systems and Computer Networking students a Visual Programming environment that would allow Information Systems and Networking students to gain an understanding of programming fundamentals without taking programming concepts further in their degrees.

This move resulted in improved SPOM results but they are still consistently below Faculty and University averages (in some areas). (See Figure 1 for an overview.)

In an attempt to improve the student learning experience, an immersive learning style has been adopted.

2. Immersive Learning

The mode of delivery that we have developed attempts to integrate presentation and practice so that learning is immediate and based on material just presented. This reflects the integrated, experiential, active and focussed practices central to immersive learning.

As Ausubel, [1] says, “A primary process in learning is subsumption in which new material is related to relevant ideas in the existing cognitive structure on a substantive, non-verbatim (sic) basis. Cognitive structures represent the residue of all learning experiences; forgetting occurs because certain details get integrated and lose their individual identity.”

¹ Paul.Sant@beds.ac.uk

² Des.Stephens@beds.ac.uk

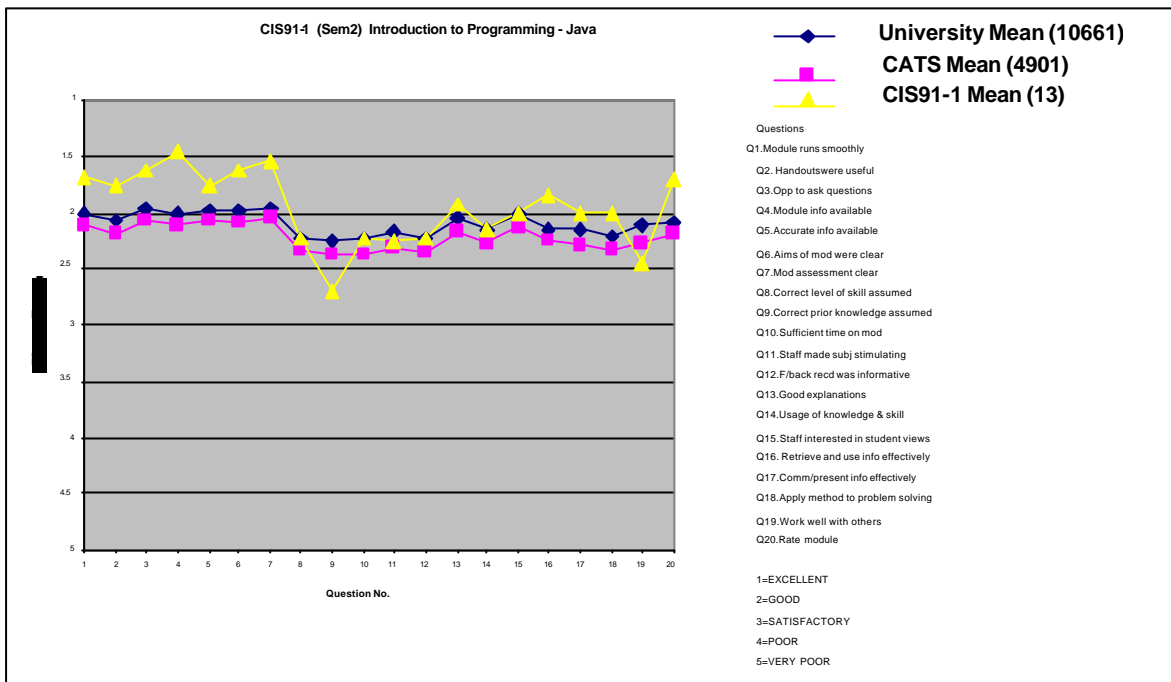


Figure 1 – SPOMS result for CIS91-1

Further, we encourage the students to work with colleagues in the laboratories and to engage with staff on their immediate problems. We do this in the light of findings by Lackney [2] that

“The brain develops better in concert with other brains – intelligence is valued in the context of the society in which we live.”

The phrase “immersive learning” has been described as [3]

“The theory of distributed representation has a profound implication for pedagogy, as it suggests that learning (and teaching, such as it is) is not a process of communication, but rather, a process of immersion. Put loosely, it suggests the idea of teaching not by telling or even demonstrating but rather through the creation (or identification) of an environment into which a learner may be immersed.”

This is often taken to mean that we learn through practice in working on something in its “natural” context - in programming this would be as part of software project team. In our context this means encouraging students to work with others, not only in the laboratories but also outside of the University environment (but supported if necessary through VLE interactions).

An extension of this practice has been widely reported in the Virtual Reality Community, see for example, [4,5].

The pedagogic design that we have evolved consists of moving the lecture from a lecture theatre into the computer laboratories and combining the lecture with demonstration, practical questions, examples and slides as well as discourse. This has been achieved by linking the presentation into 3 laboratories with projection and audio. A wireless microphone enables the presenting lecturer to roam between all laboratories engaging directly with students.

The staffing ratio enables 1 academic and 1 demonstrator per room of 30 students including the presenter. The presenter is changed during the session in order to provide variety. At the end of the presentation period a series of exercises are available for students to work on and submit for immediate registration of engagement.

The registration of engagement is recorded and forms part of the practical summative assessment marks, albeit a small percentage (12.5% of overall module marks). Note the registration of engagement, denoted by significantly attempting exercises, is not a qualitative record of programming skills. The practical assessment also requires the submission of a selection of documented exercises with some evidence of research.

In the next section we discuss the students perceptions of our immersive teaching method. The results shown were obtained via an online questionnaire which was distributed using the University of Bedfordshire's VLE, BREO. We discuss the general perceptions that students have, along with a comparison of immersive teaching and other forms of teaching that students are subjected to (both currently, and via previous experience). The results of this initial student feedback suggest some promising results.

3. Student reactions

In this section we discuss the results of a questionnaire that was distributed to students enrolled upon the Introduction to Programming module at the University of Bedfordshire.

Each of the students studying the CIS91-1 module is also registered for other Level 1 modules. This allows us to compare and contrast the immersive mode with other more traditional teaching modes (for example, the usual Lecture Practical based approach).

The sample size used here was a class of 57 students (this represents the Level 1 cohort of the University of Bedfordshire's Computing related degree programme. This excludes those students who are undertaking Information Systems degrees as these students study Visual Basic.

Let us start by briefly describing how we set up our questionnaire. In order to make sure that we received feedback relevant to our research we divided our questionnaire up into three sections. Firstly, we wanted to know how many students had been involved in the immersive teaching mode previously. Therefore the first question was to obtain information about previous experience of immersive teaching. Once we had established the prior experience of the student cohort we then needed to know how their experiences of immersive teaching compared to that of other courses, and a general picture about how effective immersive teaching was. In the final section of the questionnaire we asked the students about their own experience of immersive teaching with respect to teaching Java. Overall the questionnaire consisted of a set of ten questions and the structure of this questionnaire thus allowed us to obtain some interesting results which we shall now describe.

One of the first questions we asked students was "How familiar are you with the immersive (mixed lecture, demonstration, practical and assessment) mode of teaching". The results of this were very positive (and surprising). 94% of the students asked said that they had an experience of immersive teaching. Of this 94%, 53% were very familiar. This was quite surprising as we thought that many of the students would be unfamiliar. However, this may be explained by the fact that many of the University of Bedfordshire's students enter from Further Education colleges and they may well use the immersive mode of teaching on a regular basis. Figure 2 shows the breakdown of results.

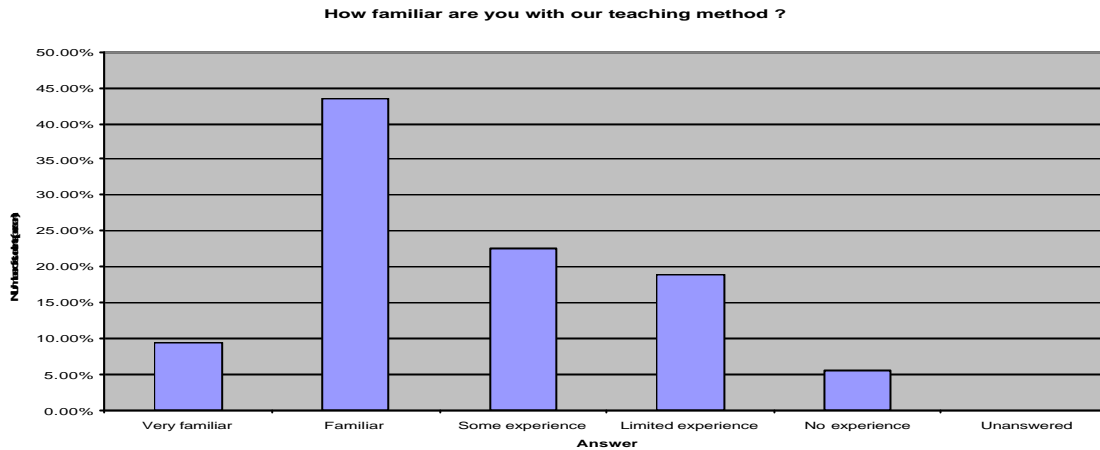


Figure 2 - Results of Question 1

The next two questions related to a comparison of the immersive mode of teaching with other forms of teaching (at Level 1 at the University of Bedfordshire). The results were positive. When we asked students to rate the immersive teaching method against the other forms of teaching that they have experienced, approximately 57% said that the immersive mode was better than the other forms of teaching that they had experienced. This is a positive result and shows that integrating demonstrations and practical's into a lecture enhances the students experience. Figure 3 shows how immersive teaching compared to other forms of teaching.

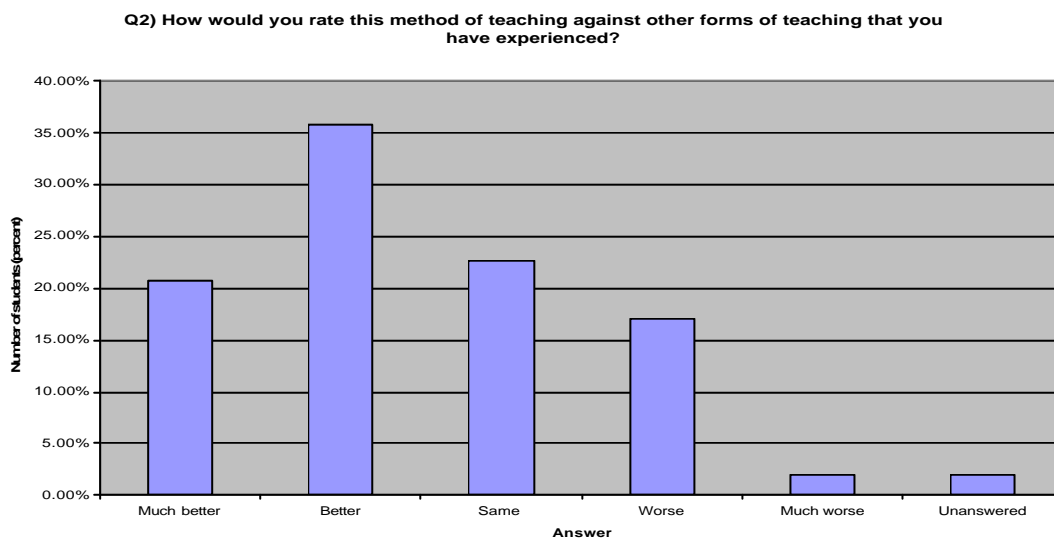


Figure 3 - Comparison of immersive teaching against other forms of teaching

In addition to comparing the immersive mode against other teaching methods, we also wanted to compare the students experience across the current forms of teaching that they are exposed to on the other courses that they are studying. The results that we obtained were very positive. Of the students surveyed 64% felt that the immersive teaching that they experienced was better (or much better) than the other forms of teaching to which they have been exposed during their first year of study at the University of Bedfordshire. This could be due to the fact that students are not only provided with knowledge, but also encouraged to practice (via class problems) the material to which they have been exposed, during the lecture. This allows them to apply their knowledge to particular problems, thus possibly enhancing their learning experience.

As well as having a cross comparison with other modules, we wanted to find out whether teaching Java in an immersive setting was having a positive effect on the students learning experience.

In order to determine how effective the immersive teaching in Java we asked a series of questions which asked about the students perception of immersive teaching, whether students felt that the immersive teaching mode improved their understanding of Java and whether immersive teaching was beneficial when it came to assessment.

In terms of student perception with respect to immersive teaching, 55% of the students surveyed said that the immersive mode of teaching was good/excellent. To back this result up when students were asked whether the immersive form of teaching had greatly improved their understanding of Java roughly half of the students surveyed agreed that immersive teaching had improved their understanding.

In terms of assessment, 52% of students felt that this method of teaching would enhance their performance in the assessments of the CIS91-1 module.

Since this is the first time that the immersive mode has been applied to learning Java, one of the important aspects that we needed to consider was whether the length of the session (4 hours, 2 hours of which are lecture, demonstration and practice, and a further two hours of assessment) was having a positive or negative effect on the learning experience of students.

As part of our questionnaire we included a comments question in which students could tell us of their experiences. The majority of comments were positive, but one area which was highlighted by students was that the length of the session was too long. Therefore, we are currently investigating whether breaking the session up would improve this aspect for the students. It may be that if the session is split into two sessions (maybe one at the beginning of the week and one towards the end) that the students learning experience is improved. This is an area of further investigation and we hope to report on this aspect in future research.

In the next section we draw some conclusions and suggest some further work.

4. Conclusions and Further Work

It would seem from the results above that immersive teaching has had a positive impact upon teaching Java at the University of Bedfordshire. These results provide a positive response to immersive teaching in the Higher Education sector, but obviously immersive teaching may not be the best form of teaching for other courses within computing, or indeed within other disciplines. However in Java it seems that immersive teaching is a highly successful method.

We hope that we have laid the foundations for the successful implementation of immersive teaching with respect to Java and we hope that this form of teaching can be applied to other areas and disciplines in the near future. The full results of the questionnaire can be seen in the attached appendix.

As part of our further work we will take our findings from our initial research (including feedback from students) and see whether altering the application of our immersive teaching mode will have a (positive) impact on the student learning process.

References

[1] Ausubel, D. (1963). *The Psychology of Meaningful Verbal Learning*. New York: Grune & Stratton.

[2] Lackney, J.A, *Design Principles Based on Brain-based Learning Research*, <http://www.designshare.com/Research/BrainBasedLearn98.htm> (accessed on 19th November 2006).

[3] Downes, S. (2005). <http://www.downes.ca/> (accessed on 19th November 2006).

[4] Dede, C. (2005). Planning for neomillennial learning styles. *EDUCAUSE Quarterly*, Number 1.

[5] Roussos, M. et al. (1999). Learning and Building Together in an Immersive Virtual World. *Presence* volume 8, no 3.

Appendix: Questionnaire Results

Results of CIS91-1 Survey on effect of immersive teaching styles on learning the concepts of Java

Q1) How familiar are you with our teaching method (mixed lecture, demo, practical and assessment) ?

Answer	Percentage
Very familiar	9.43%
Familiar	43.40%
Some experience	22.64%
Limited experience	18.87%
No experience	5.66%
Unanswered	0%
	100.00%

Q2) How would you rate this method of teaching against other forms of teaching that you have experienced?

Much better	20.76%
Better	35.85%
Same	22.64%
Worse	16.98%
Much worse	1.89%
Unanswered	1.89%
	100.00%

Q3) How would you compare this form of teaching with the lectures (of your other subjects) this year?

Much better	18.87%
Better	45.28%
Same	22.64%
Worse	11.32%
Much worse	1.89%
Unanswered	0%
	100.00%

Q4) In terms of your learning experience, how would you rate this form of teaching?

Excellent	11.32%
Good	43.40%
Average	35.85%
Poor	9.43%
Terrible	0%
<i>Unanswered</i>	0%
	100.00%

Q5) How would you rate the following statement: "I found this mode of teaching was stressful / overloaded my concentration"

Strongly Agree	7.55%
Agree	18.87%
Neither Agree nor Disagree	30.19%
Disagree	28.30%
Strongly Disagree	13.21%
<i>Unanswered</i>	1.89%
	100.00%

Q6) How would you rate the following statement: "This mode of teaching is unfair to students who are unable to attend sessions on a regular basis"

Strongly Agree	7.55%
Agree	18.87%
Neither Agree nor Disagree	33.96%
Disagree	28.30%
Strongly Disagree	11.32%
<i>Unanswered</i>	0%
	100.00%

Q7) How would you rate the following statement: "This mode of teaching has significantly improved my understanding of programming in Java"

Strongly Agree	11.32%
Agree	39.62%
Neither Agree nor Disagree	35.85%
Disagree	5.66%
Strongly Disagree	7.55%
<i>Unanswered</i>	0%
	100.00%

Q8) Are you happy with this mode of teaching ?

Yes, I am very happy	16.98%
Yes, I am happy	41.51%
I am indifferent	26.42%
No, I am unhappy	11.32%
No, I am very unhappy	3.77%

Unanswered 0% 100.00%

Q9) When it comes to assessment do you feel that this mode of teaching is

Very beneficial	13.21%
Beneficial	49.06%
Neither beneficial nor unhelpful	30.19%
Unhelpful	7.55%
Very unhelpful	0%
Not Applicable	0%
<i>Unanswered</i>	0%
	100.00%

Q10) How would you rate the following statement: "All my lectures should be delivered in this way"

Strongly Agree	15.09%
Agree	24.53%
Neither Agree nor Disagree	33.96%
Disagree	18.87%
Strongly Disagree	7.55%
<i>Unanswered</i>	0%
	100.00%

Teaching Ruby on Rails
Dr Bruce Scharlau
Computing Science Department
University of Aberdeen
Aberdeen, AB24 3UE
scharlau@csd.abdn.ac.uk

Abstract

This paper considers the teaching of the object oriented scripting language Ruby within the Web framework of Rails to university students. This paper argues that teaching Ruby on Rails is a good choice for teaching students appropriate development habits through its rigorous application of the model-view-controller design pattern in the application code. Furthermore, productivity gains from using the Rails framework means that students can tackle more challenging issues when developing web sites. This is because most of the defaults for Rails work for everybody so developers can focus on the business logic and not the object/relational mapping of the CRUD details.

1.0 Introduction

Ruby on Rails [1] has caused a storm of interest in the last two years since it was first introduced [2]. Somehow this seems to have been overlooked by students, who never have heard of it by the time they arrive in level three (under the Scottish system).

This paper explores the reasons for teaching Ruby on Rails to undergraduate students. The next section details the technology. The third section describes how it is being taught, and the last part covers lessons learned from the experience to date.

The Ruby programming language was released to the world from Japan by Yukihiro Matsumoto in 1995. Ruby is an object orientated scripting language. We can write classes to model real world objects and have them interact in a way familiar to us [1,3]. The same as with Java, for instance, we can have our sample application model ships that go on cruises filled with customers booked into their reserved cabins.

Rails is a framework for Ruby-based web sites that is the result of D. H. Hansson extracting the core from the work he had been doing on the Basecamp application for 37 Signals. It was released in July 2004 [3]. Rails provides the glue for our Ruby coded sample application to display the correct page showing on which cruise our customers are booked.

Rails extends the object oriented nature of Ruby by placing its applications into the Model View Controller design pattern as shown below in Figure 1.

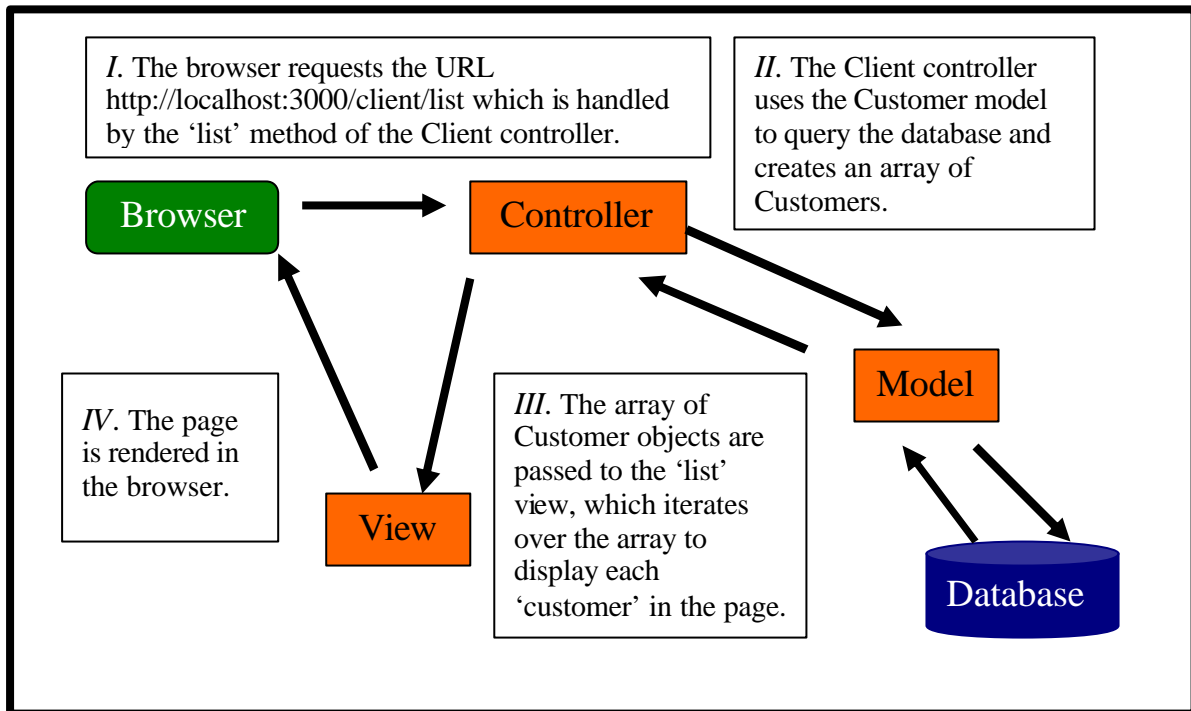


Figure 1: Model View Controller in a Rails application

To make the relationships between model, view and controller more apparent, each Rails application is laid out in a strict format so that the developer knows the 'best' place to put the appropriate code. Methods dealing with the business logic go into the controllers. Callbacks for the models, and details of their validation go in the model files. Anything affecting the presentation of the results that have been gathered by the method in the controller, go into the view. Rails follows, in other words, the DRY philosophy of 'do not repeat yourself' [4]. Find the best place in this framework for the bit of code you're working on, and put it there.

As with C# in ASP.NET or Java web applications, Ruby on Rails provides code with business logic behind the page in the browser. However, it is easier to create maintainable applications with Rails than with C# or Java. The Rails framework means that students do not have to be taught how to piece the parts together from scratch for an application following the Model View Controller design pattern. With Ruby on Rails the student's application has to work within this pattern.

3.0 How we Teach Ruby on Rails at Aberdeen University

Students at the University of Aberdeen on the BSc Degree (Single Honours) in Internet Information Systems take the module 'Web Design and Administration' in level three. This module replaces the Formal Languages and Compilers module that the other BSc degree Computing students offer, and is the point at which the streams diverge. At the same time they are also starting their year long software engineering group project, which for the first term focuses on design patterns and software engineering artefacts. In the second term the students deliver their working group project. By this time all level three students have done two years of Java and one term of C. The module also has the feature that more than half of its students are Erasmus students, who come with a variety of backgrounds and abilities.

About one-third of the 'Web Design and Administration' module focuses on teaching a web application scripting language. A scripting language was deemed appropriate as they study web application development with Java and C# with ASP.NET in the following term. Alongside this the module also covers the Web standards of XHTML and CSS, Information Architecture, and administrative issues around web sites such as content management systems, security, privacy and mobile web site design.

Since the module started three years ago, PHP was never a satisfactory choice for a language in this module. While PHP is easy to pick up, it also leads to messy and hard to maintain sites if the developer is not strict about the structure for the code produced. Many students repeat code such as database access and menus in many places instead of using page includes and function files. If students do not enforce their own rules about code maintainability from the start of their application building, then they have problems. PHP does not penalise people for messy code, and it can quickly lead to a maintenance nightmare. In addition, there are always a number of students, who have already learned PHP, and they lose interest in this part of the course.

A Rails application is the opposite. It is new to students, and excites interest because of its different development cycle. A new project is created with a command such as 'rails travelagent' which creates a project with the model-view-controller directory structure, and the developer has to work within this as she develops the project. Controllers and models are then added with further commands such as 'ruby script/generate scaffold -s Customer Client' which, in this example, creates the Customer model and ties it to the Client controller, and generates scaffolding pages associated with the views of the controller methods. With a third command we can start the integrated Web server and then, assuming that we've already configured the database, view a list of customers and perform basic create, read, update and delete operations on them.

As the students move through the Ruby on Rails part of the module they work with the ongoing 'travel agent' example [5]. This example is an understandable scenario to students and works on a number of levels. It provides a simple scenario to be explored, which can be made as complex as time allows because of the relationships between the models. It also shows that while the basic CRUD operations can be auto-generated, the developer needs to know what business flow is required by the Web site in order to build appropriate navigation systems.

After the four two-hour Ruby on Rails practical sessions the basic cruise ship site lets travel agents book cruises for their clients. The first sets out the basic controllers and models so that at the end the students have all of the basic CRUD operations available on the ships, cabins, customers, cruises, credit cards, addresses, payments and reservations. The second puts the relationships between the models into place, so that, for example, each customer has one address and many credit cards. The third practical adds a common menu to the template pages, and explores sessions, while the fourth looks at filters and helpers in the application.

The basics of Ruby and Rails are introduced as the class progresses from the first practical session which builds the basic travel agent site. They start with what they need to know to generate and edit the files, and become familiar with instances and objects. In the second session they work with Ruby to iterate through arrays as shown in the example below, and how to create and persist objects. In the third session they learn how to handle exceptions in Ruby, and the fourth covers helpers and filters.

Rails automatically does object/relational mapping for the developer using a default system where it expects table names to be the plural of the model class name. Rails developers do not need to work with SQL directly. Instead Rails generates the SQL query at runtime. Therefore the student becomes more familiar with working with objects, instead of, as with PHP, having to map back and forth between SQL results and the variables in the code.

With Rails handling the O/R mapping the students progress further in their application building during practical sessions. This allows time to explore more complex issues such as the relationships between models and the association of these classes from a software engineering perspective.

The parent-child relationship between objects and table extends this. It becomes transparent in Rails as the relationships are stated in the code for the models so that Rails knows that each Customer has one Address and many Creditcards. There is no need to also write this relationship in SQL unless it is to ensure that any other application using the tables also follows the same rules.

In our case we only need the basic model rules in place. On the Creditcard side we need to declare that

```
class Creditcard < ActiveRecord::Base
  set_table_name "csd123_credit_card"
  belongs_to :customer
end
```

And on the Customer side we declare that

```
class Customer < ActiveRecord::Base
  set_table_name "csd123_customer"
  has_one :address
  has_many :creditcards
  has_many :reservations
end
```

It now becomes easy to display all of the credit cards used by a customer without having to write any SQL. Instead it is a simple matter of looping through the instances to display the values as shown below:

```
<b>Credit Cards:</b> <br>
<% @customer.creditcards.each do |cc| %>
<%= link_to 'Edit Card Details', :controller => 'card', :action => 'edit', :id => cc.id %>
<%= cc.number %><br>
<%= cc.name_on_card %><br>
<%= cc.organization %><br>
<%= cc.exp_date %><br>
</p> <% end %>
```

The code above is interpreted like this: The line `<% @customer.creditcards.each do |cc| %>` starts the loop and means 'for each credit card of the customer, which we'll call cc'. The next line creates a link labelled 'Edit Card Details' and links to the edit method in the card controller passing in the parameter of the creditcard id. The following lines display the useful instance details of the credit card, and then the loop is ended with `<% end %>`. Given that each customer has many reservations per the O/R mapping, we can display them in a similar fashion.

Students should better appreciate frameworks and how their use speeds development after working with Rails. The framework forces specific rules to be followed, which is not always clear to the students at first. Over time students should understand the model view controller pattern as it also appears in other classes and they appreciate it as a transferable framework. This is more useful than what they would learn if they were using PHP and SQL.

The goal is to have the student understand that Web development should be a matter of creating models of the object and then using these as data transfer objects, whose values we can then display in the web page. The only difference being whether we need to send a single instance, or whether we need to send a collection of objects, which we then iterate through to display values.

While the travel agent application remains incomplete, it does show how the different parts of Rails and Ruby help to build a complex application. It shows that by taking away the low level work, students can dedicate more time to the business logic of the application. Students can determine the best place to take payment: whether it is correct to tie this into the reservation process, or to deal with

it separately so that you can take multiple payments for each reservation. We can also use the example as a backdrop for learning AJAX and for working out best practices for web sites viewed from mobile phones, as well as how to generate RSS feeds.

4.0 Lessons Learned

This was the second year that we've used Ruby on Rails. Last year it was done on a 50:50 basis with PHP so that half the practicals were done using PHP and then the students switched and did the same application using Ruby on Rails. That did not seem to make much sense though as one practical session with Rails brought the students to the same point in the application which took three previous practical sessions with PHP. This was a result of Rails doing the 'grunt work' of the CRUD details.

This year with more time to work with Rails has meant that the practicals can provide more scope to explore the complex issues of relationships, which were only skimmed over last year. It was hoped too that the further development of Rails would have resolved a thorny issue about table names and model names in Rails. This hasn't happened, and is unlikely to be resolved [6], which means more work in teaching Rails.

Rails makes many assumptions about the applications it is building so that 'most of the time the default options work' [4] and developers can ignore them. This is fine in most situations. However, it causes problems in a classroom situation when students share a database.

A student using a shared MySQL database takes longer to develop her models and controllers because a few components need to be put into place to avoid the group overwriting, and deleting each other's tables. They each need to prefix their table names with their usernames, and to then tell their models to use these specified table names. Lastly, they need to turn off the 'pluralise table names' rule. After these preparations the student can now create their scaffolding and models attached to controllers with a command like 'ruby script/generate scaffold -s Customer'.

However, usually in the shared situation, instead of seeing Rails generate visible scaffold pages, which the students can edit, they see error messages explaining why the scaffolding was not generated. There is now a disjuncture between their building of an application in theory, and how it works in their own experience. Although the correct files can be provided to them, it still remains a bit of a let down for the students, as they implement the kludge of downloading the files and dropping them into their application instead of generating the files themselves.

A better solution would be to provide each student with a database to use so that they can use Rails as they would if they were developing on their laptop, or for a firm. Making sure that this can be done means extra work, but should resolve this one crucial issue.

The travel agent site also needs to be spread over a larger number of practical sessions. Some students felt that we moved too quickly and that it would be better to go over the Ruby more slowly, while introducing Rails in the first practical.

The last point is that the curriculum needs to be reviewed periodically to see that what is taught in the classroom is still relevant, and whether something better might be appropriate and available. In this case Ruby on Rails was a better choice than PHP for the web scripting language. It provides teaching materials that are better suited to learning about software development in general, and web site development in particular, and which reflect a modern agile orientated software engineering practice similar to what the students will encounter after they graduate.

References

[1] <http://www.rubyonrails.com/>

[2] <http://www.onjava.com/pub/a/onjava/2005/11/16/ruby-the-rival.html?page=1>

[3] David A. Black (2006) *Ruby for Rails*, Manning

[4] Dave Thomas, David Heinemeier Hansson (2006) *Agile Web Development with Rails*, 2nd Ed. Pragmatic Programmers.

[5] Based on one found in Richard Monson-Haefel (2001) *Enterprise JavaBeans*, 3rd Ed. O'Reilly.

[6] See the note from Hansson at <http://dev.rubyonrails.org/ticket/913>.

A Step Back from Coding - An Online Environment and Pedagogy for Novice Programmers

Andrew Scott, Mike Watkins and Duncan McPhee
Faculty of Advanced Technology – University of Glamorgan

Abstract

This paper proposes the use of an interactive flowchart based visual problem-solving tool and code generator to address issues faced by novice programmers. Many novice programmers are demotivated while undertaking their initial steps in programming due to issues associated with the syntax and semantics of the programming language, the skill requirements of problem solving and program design and the development environments they are expected to use. The use of the tool focuses on using flowcharts to develop visual solutions to basic programming problems from which syntactically correct program code is generated. The animation features and interaction between the visual and code representations reinforce student understanding of both the visual solution and program statement flow. The application of the tool is coupled to a classroom pedagogy based on the principles of scaffolding support and their combined use aims to increase student comprehension and maintain student motivation when engaging in the introductory levels of programming.

1 Introduction

It is a well established fact that many students find introductory programming concepts difficult to master [22]. Novice programmers must contend with many new things: the development environment, the syntax and semantics of a programming language, an understanding of programming concepts and any paradigm specifics. They also need problem solving skills and strategies in order to solve programming problems effectively. With so many new things to learn, it is easy for students to become overwhelmed and de-motivated. Our research has identified that many programming beginners possess weak problem solving skills and have a deficiency in their algorithmic problem solving abilities. In this paper we propose a programming tool and associated pedagogy designed for the first term of an introductory programming course. Our aim is to reduce the impact of complex development environment and language syntax and focus on overcoming weaknesses of algorithmic problem solving and subsequent development of source code. Addressing these issues is pertinent to programming in any paradigm or language and forms an excellent foundation from which the student may graduate on to more complex concepts. This paper follows on from our earlier work presented at the JICC10 [19].

2 Novice Programmer Problems

There is little doubt that students have problems with introductory programming. Roberts states 'Programming courses are generally regarded as difficult and often have the highest drop out rates' [17]. For all but the best students, introductory programming courses can prove to be very confusing, especially during the first term. This claim is backed up by the large amount of literature devoted to the numerous problems of novice programmers, that identifies the following issues:

- ? The syntax and semantics of the programming language;
- ? Program design and problem solving;
- ? Development environment;
- ? Paradigm specifics such as objects,

The syntax and semantics of a programming language can have a profound effect on the performance of novices. McIver [11] states that trivial syntax errors may in fact impede learning, as they distract students from the fundamentals of programming and problem solving. Empirical evidence suggests that students who are struggling with syntax often make things worse when attempting to correct errors. Payne and Myrers [13] highlight many syntactic and semantic difficulties in the languages used by novices. Many of these prevail in the Java language as pointed out by Biddle [4]. In a three-year study, Garner et al [8] discovered that minor syntactical errors persisted as the most common cause of problems for novice programmers in both the stronger and weaker students throughout the duration of an introductory programming course. McIver [11] states that novices are forced to deal with syntax and the individual elements of their algorithms, sometimes to the exclusion of the broader picture of the problem at hand.

A study by Garner et al [8] indicates that problems of program design come second only to the problems of syntax and are more prominent than problems relating to the comprehension of individual language constructs. The problems of program design concur with Sohporer and Soloway [21] who

claim syntactic and semantic issues are not the major difficulty. They state that, 'students have difficulties in putting all the pieces together' and 'many problems arise from structure composition problems'. Winslow [27] reached a similar conclusion stating, 'Study after study has shown that students have no trouble generating syntactically valid statements once they understand what is needed. The difficulty is knowing where and how to combine statements to generate the desired result.' Howell [9] went as far as to say, 'Students no longer master problem solving skills, as they are no longer emphasized in elementary education. Students have become so weak in the development of learning strategies and problem solving skills that it adversely affects independent and meaningful learning in all disciplines.' This concurs with Robins et al [18], who provide a comprehensive review of research related to the current problems of learning and teaching programming. In this report they conclude that the difference between effective and ineffective novices relates to strategies they employ when applying their programming knowledge. These views imply that in general, novices have weak problem solving skills. While novices may attain knowledge of the syntax of individual components, it is clear that novices have significant problems putting all the programming constructs together in order to express a solution.

Traditionally, many introductory courses immerse their students in fully featured professional development environments. Muller and Hosking believe that the chief difficulty faced by novice programmers in using the Eclipse IDE is the learning curve necessary to just get started [12]. Reis and Cartwright [15] state, 'Professional IDEs for Java work well in advanced courses, but they are poorly matched to introductory courses because they deluge beginning students with a complex array of features.' A professional IDE will be counter-productive to introductory programming, as the vast array of visual and functional complexity may overwhelm and intimidate a novice. Also, the syntax error messages generated by these environments are often uninformative and sometimes misleading.

A modern trend in the teaching of novice programming is the objects first approach [2,1,16]. Tools such as BlueJ [2] have made the objects first approach more compelling. As a result, several universities have adopted this approach. By its nature, the learning curve of an objects first approach is much steeper than the traditional imperative approach. This point is reinforced by Ramalingham and Weinbeck [14]. Rist [16] warns that object oriented (oo) programming is not different; it is more, because oo adds overheads such as class structure. Object oriented programmers must learn the basics of the imperative style, including data types, assignment, branching, looping, and functions, as well as the oo features [14]. Therefore, we can assume that objects first novices will encounter many of the same difficulties (plus more) as students studying from the traditional imperatives first approach. For this reason, we feel that objects can only be tackled once the basics of sequence, selection and iteration have been mastered.

3 Flowchart Visualisation of Computer Programs

Ramalingham et al [14] state that Programming is a highly cognitive activity that requires the programmer to develop abstract representations of a process in the form of logical structures. They go on to say that if a novice can attain an accurate mental model, it can positively affect their success in programming. They also state that developing mental models should remain a goal in introductory programming courses. This view is consistent with Winslow [27], who asserts that models are crucial to building understanding. There is a large body of evidence to support the notion that carefully designed visualisations can aid a novice user to develop an accurate mental model, for example [10,23]. Computerized aids that reinforce mental models of the programming process are more appropriate for novice programmers than commercial environments [20]. ICT has regularly used visual representations of its systems at all levels. Flowcharts have traditionally been used to visualise programming structures and are an excellent form of visualisation for novices. They can be easily understood with little or no prior training and provide the novice with an accurate mental model of an algorithm. Furthermore, flowcharts aid the transition between problem specification and syntactical solution; this transition is a major cause of difficulty for novices. Westphal et al [26] point out that 'without the use of diagrams or flow charts, it is difficult for beginners, even with pseudo code to communicate the flow of a program. For example, the next instruction to follow may not always be obvious from reading pseudo code.' While flowcharts are not well suited for large and complex programs, we feel they are very appropriate for modelling simple programs and concepts for novices. It could be argued that UML activity diagrams are a more appropriate and modern representation. Activity diagrams bear a close visual and functional resemblance to the traditional flowchart and offer advanced features that surpass the capability of the standard flowchart notation, such as stereotyping,

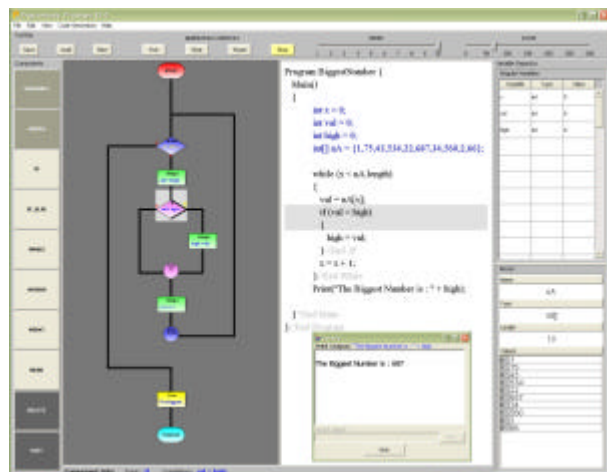
swim-lanes, split and join. These are features that are not needed by novices who are modelling simple programs [19]. Therefore, in the context of simple novice programs the activity diagram has no fictional advantage over the traditional flowchart. However, it will be of benefit for novices to familiarise themselves with this aspect of UML notation as UML may be increasingly important in their future studies of programming. We do plan to include activity diagrams in the future although currently our focus is on flowcharts.

A program that allows the user to construct a program visually via a flowchart based representation will provide the novice with an effective mental model whilst placing a greater emphasis on the problem solving tasks of programming. This may help the novice programmers overcome their problem solving difficulties of abstracting a problem into a solution. With current technology, computerized visualisations can be dynamic rather than static in nature. Tudoreanu [23] describes program visualization, also termed algorithm animation, as ‘a form of presenting the execution of a running computation through the use of animated graphical displays.’ Ben-Bassat Levy et al [3] state that the main benefit of animation is that it offers a concrete model of execution that all but the best students need in order to understand algorithms and programming. It is logical to assume that by animating a flowchart we may enhance its effectiveness by visualising program flow and the interactivity between program components. Thus, we would enhance its efficacy as problem solving aid.

Recently there have been an increasing utility of flowchart based programming environments. Raptor [5], Flow Chart Interpreter (FCI) [7] and Flint[6], are flow chart based visual programming environments designed for novice use. Both Raptor and FCI utilise animated flowcharts with the basics of sequence selection and iteration underpinning their design. In a study, Carlisle et al [5] found that the average grades of students using Raptor were significantly higher than those using the Ada or MATLAB development environments. Studies by Crews[6] displayed similar results. Raptor, FCI and Flint provide no facility to view or generate computer code in its traditional text based form. While we don’t want to burden our users with the complexities of writing code, we feel it is important for novice programmers to familiarise themselves with the structure of program code. Completely sheltering the user from any exposure to program code will cause a steeper learning curve later on when they graduate from the tool. It is also important for the user to observe the relationship between a flowchart and the textual code structure that a subsequent design would likely produce. This will enable the users to relate their mental model to program code. As final example, the SFC Editor [25] does provide the visual duality of code and flowchart, although its lack of execution or animation features makes it less suitable as a novice debugging and problem solving aid. To run a program, users must export the generated code to another environment. Using two environments to solve one problem introduces unnecessary complexity into the learning environment.

4 The Progranimate Programming Aide

We have implemented a tool that allows the user to focus on improving their algorithmic problem solving skills. This has been achieved by minimising the necessity of writing complex and confusing syntax and reducing the learning curve associated with professional development environments, which often cloud the problem solving processes at the heart of programming. The basic concept behind the tool is to visualise the programming task via flowcharts and automatically generated computer code. Our tool combines and elaborates on many of the ideas discussed in the last section. The user interface is split into three areas of visualisation: the



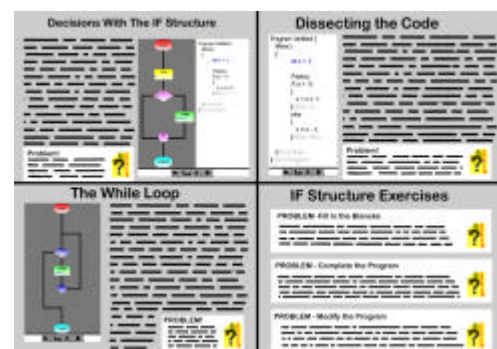
flowchart, the generated code and the variable inspector. In Progranimate users can create a program by interacting either with the flowchart or the generated code. They do not have to trouble themselves with entering large amounts of complex and confusing syntax. As pointed out in section three, we feel it is beneficial for students to familiarise themselves with computer code. A feature of our tool is the automatic generation of code in a variety of languages, including Java. This is generated on the fly as each flowchart component is defined and added. A program is constructed by choosing a component

from a palette and then clicking on the relevant part of the diagram or program code to add it. A window will then pop up where the user is able to enter a relevant expression for this component. For example, “Hello” + name for a print component, $x \geq 2$ in the case of the If and While components, or $x = x + 1$ in the case of the assign component. Once the user’s entry has been validated, the flowchart is updated tidily, and the relevant lines of code are generated and inserted appropriately. The component palette allows the user to create a program from the following components. Print – Performs the output of program data to the console window. Read – Allows user entered data to be input from the keyboard into variables and array elements. Assign – Assigns the value of one variable/array element to another. It also facilitates the assignment of mathematical expressions, conforming to the normal rules of precedence. If and IfElse - These structures form standard decisional constructs and redirect the flow of program based on the result of the conditional expression they contain. While - Facilitates a basic looping construct that relies on the result of the conditional expression it contains. Program variables and arrays are also added, using the palette. Once created, the variable / array declarations will appear within the generated code but not the flowchart, as they play no part in visualising program flow. The program variables and arrays are visualised in the variable inspector. This allows the user to view, modify or remove the program variables and arrays. It also allows the user to view the changing state of program variables and arrays during run time. We believe that the array visualisation may help to overcome much of the confusion associated with the novice use of arrays. For example, many novices get confused when working with zero based arrays because the last element of an array of size ten is stored in index nine. It is also not uncommon for novices to confuse an array element’s subscript with its value. By visualising an array’s size, element index and element contents, it is hoped that we may prevent much of this confusion by making things very apparent. We have also attempted to overcome the confusion often associated with the cryptic and misleading error messages generated by professional environments. Error messages generated by the tool are designed to point the user directly to the source of the problem in plain English. However, many common errors have been eliminated due to the code generation features. Created programs and program settings can be saved to file and be loaded at any time. An undo feature is also included as novices will be prone to making mistakes.

Progranimate provides the facility to animate its computer programs. The animation works by tracing the program flow as it executes, highlighting each flowchart component and relevant line of code in turn from start to end of the flowchart. As each program instruction is encountered, the program flow or values of the variables and arrays elements in the inspector change appropriately. Users can run an animation at a desired speed, pause a running animation or control each step of the animation manually. Our research has shown us that novices have trouble abstracting a problem into a solution and a solution into a program. One cause for this is a difficulty in understanding the interaction between program components. Many Novices have trouble combining the concepts of data types, assignment, branching, and looping to derive an effective solution. It is our hope that the animation feature will provide novices with an accurate mental model of program execution and make explicit the interaction between the program components. This will also help the user to develop effective debugging strategies.

4.1 WWW Integration

As well as working as a standalone tool, Progranimate can now be delivered to users via Java Web Start. This allows the tool to be launched by any computer with Internet access and Java 1.5. This removes the hassle of installation and makes it easier to access the tool outside the classroom. It also makes it possible to launch files contained within a website directly into Progranimate, thus making the tool ideal as the centrepiece of an online tutorial. Another feature of the tool is its ability to be deployed as an applet. The applet can load files stored locally or contained on a web server. It provides play, step, pause and stop buttons for the user to run an animation. It also has full access to the any of the visual elements and features available to the standard application. This gives the web master the ability to customise the applet by enabling and disabling its visual elements, changing its dimensions, zoom level, animation speed and program language. This further enhances its usefulness as an online tutoring aide, as it can be used to visualise and animate programming concepts directly within a web page. For example a tutorial page may



discuss a concept such as a while loop and use the applet to demonstrate this concept by animating it within the page presenting a flowchart, program code or both representations. The foot of the page could then pose a relevant programming problem for the user to solve. The users may then click on a link to launch Progranimate. The tool would then present either a partially completed solution or a blank canvas so that the user could attempt or observe the posed problem solution. These features make the tool ideal as the centrepiece on an online tutorial.

5 Teaching Pedagogy.

In Vygotsky's [24] social development theory, he defines an instructional strategy commonly known as scaffolding, a term which was actually coined by Wood, Burner and Ross [28]. In this context scaffolding is a temporary supportive structure that assists a student to accomplish a task that he or she could not complete alone. This supportive scaffolding is gradually removed as the learner becomes competent at a given task or concept. Eventually, the aim is for the student to be self-sufficient and able to handle the task or use the concept independently of any supportive scaffolding. Vygotsky describes scaffolding as adult guidance or peer collaboration. However, in the technological age, this supportive structure may also encompass computer-aided instruction. This means that a pedagogy based on these scaffolding theories may be an appropriate choice for an online learning environment encompassing programming. In Vygotsky's theories the zone of proximal development (ZPD) defines the gap between a student's independent problem solving skills and the problem solving skills they achieve with help and guidance. To perform effective scaffolding the tutor (or supportive instructional environment) should stay within the ZPD, this being one-step ahead of the student's current abilities. If the tutor goes outside the bounds of the ZPD (too far ahead or too close to the students current abilities) little or no learning will occur. In all cases the tutor should act as a scaffold, providing the minimum support necessary for a students to succeed in each task. This scaffolding pedagogy is split into four key stages.

- ? Tutor modelling behaviour for the student
- ? Student imitating the tutor's behaviour
- ? Tutor fading out instruction and support
- ? Independent problem solving ability and possible ability to provide scaffolding for less able peers

If we combine the use of an online instructional environment with a pedagogy based on the scaffolding approach, we may approach the concepts of programming in gentle steps. As we can integrate our tool into an online environment, we can use it to greater effect when teaching the basics of programming to novices. Our plan is to use this approach to further assist the users in overcoming their algorithmic problem solving deficiencies. We may then see an improvement in the problem solving skills, self-efficacy and all around competence in the programming abilities of our novices.

Our pedagogy is currently a proposal and is intended for teaching introductory programming concepts to undergraduate novices or foundation year students. It is designed to take novices from the basics of variables, data types, assignment, and decisions though to looping. However, this pedagogy could be easily extended to encompass arrays. As the students progress though the course, they will combine newly learnt concepts with the knowledge gained in previous lessons. By getting the students to continually revise their knowledge while learning new concepts, we will reinforce potentially fragile knowledge that may otherwise be forgotten. As an introduction, the first tutorial will be devoted to familiarising the users with the online environment and programming tool. We plan to deliver the each programming concept in four key stages that gradually removes the support given to the student. Vygotsky places importance on the social interaction and collaboration of learners. In our pedagogy we aim to promote in class discussion and collaboration in its first two stages. Each individual basic programming structure (sequence, selection and iteration) is covered in the four key stages of our pedagogy. These stages are based on the four stages of the scaffolding pedagogy discussed above.

Basic Programming Structures and Concept Demonstration

A basic programming structure is discussed and visualised using the online tool. Further demonstration of additional concepts such as the combining and nesting of these structures are also visualised and demonstrated. The animation features of the tool reinforce an understanding of program flow concepts in both the visual and code representations of these structures.

Guided Problem Solving

Using the tool, initial flowchart and code solutions to selected problems are presented to the class and each student workstation. These are developed by the tutor through collaborative class interaction with

students. Each student actively engages in implementing these solutions on their own work-stations. Again the animation features increase student understanding of program statement flow and behaviour.

Assisted Problem Solving

This stage involves students in applying the tool through a series of steps, where at each step the tutor and system support scaffolding provided is gradually reduced. Step one provides the student with a partial visual solution in which some of the components have missing parameters; the student needs to fill in the blanks from a supplied list of possible parameters. Step two presents students with a partially completed solution where some of the components are missing, the student is tasked with completing the program by adding components from a supplied list of components. Lastly in step three, the students are tasked with modifying the program created in step two to extend or change its functionality. These steps can be extended with additional intermediate steps where lists are not provided.

Un-assisted Problem Solving

In this final stage the users are presented with a range of similar problems of increasing complexity. This time the users are given a blank canvas and have to solve the problem from the ground up with minimal guidance from the instructor and the online environment.

This pedagogy is suitable for the first semester of an introductory course in computing. It may also suit foundation year and high school students as a valuable precursor to an introductory programming course. We feel that this will provide an excellent foundation from which the students can graduate onto more complex topics. A preliminary evaluation of the pedagogy and utilisation of the tool is planned for early 2007 with further evaluations scheduled for mid to late 2007.

6 Conclusions and Future Work.

From our research, it has become very evident that over all, students have a major weakness in their problem solving abilities, especially in the context of programming. More specifically, a key difficulty is in their algorithmic problem solving abilities of abstracting a solution design into a computer program. Complex development environments, perplexing syntax and paradigm overheads are clouding the tasks that novices struggle with most, thus adding to the confusion. By focusing the novices' attention purely on the problem solving exercise, we believe that they may better overcome this key weakness. In this paper we have presented a tool that aims to improve on the problem solving abilities of novice programmers by attempting to overcome complications of the development environment, the writing of complex and often confusing syntax, and any paradigm specific overheads. The tool allows the user to focus solely on the problem-solving task by minimising all other distractions. We have proposed to combine use of the tool with Vygotsky's scaffolding pedagogy. It is our view that this approach will enhance the novices' knowledge of the basic imperatives of programming that underpin all paradigms and as a result, improve their algorithmic problem solving abilities. This will provide a solid foundation from which the novice programmer may graduate to more complex concepts such as objects.

Further development of the programming tool is currently being conducted to advance its usefulness and flexibility with respect to use within an online learning environment and the described pedagogy. We are also plan to provide features that enable functional decomposition within user programs. The code generation features are a relatively recent introduction into the programming tool. Therefore, to assess the effectiveness of these features and the proposed pedagogy, further evaluation is needed.

7 References

- [1] **Arnold D and Weiss G**, 2001, Introduction to programming using Java: an object-oriented approach, Java 2 update. Addison-Wesley, Boston USA, ISBN 0-20-161272-0.
- [2] **Barnes D and Kölling M**, 2006, Objects First With Java – A Practical Introduction Using BlueJ - Third Edition, Prentice Hall, Harlow UK, ISBN 0-13-197629-X.
- [3] **Ben-Bassat Levy R, Ben Ari M and Uronen P**, 2001, An Extended Experiment with Jeliot 2000. In Proceedings of the First International Program Visualization Workshop, *University of Joensuu Press*, Porvoo Finland, pp: 131-140
- [4] **Biddle R and Tempero E**, 1998, Java Pitfalls for Beginners, ACM SIGCSE Bulletin, Volume 30 issue 3, *ACM Press*, New York, pp: 48-52.

- [5] **Carlisle M, Wilson T, Humphries J and Hadfield M**, 2004, RAPTOR: Introducing Programming To Non-Majors With Flowcharts, *Journal of Computing Sciences in Colleges*, *Consortium for Computing Sciences in Colleges*, University of Central Missouri, USA pp: 52 – 60
- [6] **Crews T**, 2001, Using a Flowchart Simulator in an Introductory Programming Course, *Computer Science Teaching Centre Digital Library*, *Western Kentucky University*, USA, *Online: <http://www.cstc.org/data/resources/213/Visual.pdf>* [Accessed 18/11/06]
- [7] **Atanasova G and Hristova P**, 2003, Flowchart Interpreter: an Environment for Software Animation Representation, *Proceedings of the 4th international conference on Computer systems and technologies*, *ACM Press*, Sofia, Bulgaria, pp:453 – 458
- [8] **Garner S, Haden P and Robins R**, 2005, My Program is Correct But it Doesn't Run, *Proceedings of the 7th Australasian Computing Education Conference*, *Australian Computer Society*, Newcastle NSW Australia, pp: 173-180
- [9] **Howel K**, 2003, First Computer Languages, *Journal of Computing Sciences in Colleges* Volume 18 Issue 4, *Consortium for Computing in Small Colleges*, USA, pp: 317 – 331
- [10] **Kann C, Lindeman R, and Heller R**, 1997, Integrating algorithm animation into a learning environment, *Computers & Education* Volume 28 Issue 4, *Elsevier Science Ltd*, Oxford England, pp: 223–228.
- [11] **McIver L**, 2000, The Effect of Programming Language on Error Rates of Novices, *12th Workshop of the Psychology of Programming Interest Group*, *Psychology of Programming Interest Group*, Cozenza Italy, pp: 181-192.
- [12] **Mueller F and Hosking A**, 2003, Penumbra: An Eclipse Plug-in for Introductory Programming, *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange - Eclipse 03*, *ACM Press*, Anaheim California USA, pp: 65-68.
- [13] **Payne J and Myrers B**, 1996, Usability Issues in the Design of Novice Programming Environments, *Technical Report CMU-CS-96-132*, *School of Computer Science -Carnegie Mellon University, Pittsburgh - USA*, *Online: <http://www.cs.cmu.edu/~pane/ftp/CMU-CS-96-132.pdf>* [Accessed 19-12-2006].
- [14] **Ramalingham V and Weidenbeck S**, 1997, An Empirical Study of Novice Program Comprehension in the Imperative and Object Oriented Styles, *7th Workshop on Empirical Studies of Programmers*, *ACM Press*, Alexandria –Virginia - USA, pp: 124 – 193.
- [15] **Reis C and Cartwright R**, 2004, Taming a Professional IDE for the Classroom, *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, *ACM Press*, Norfolk - Virginia - USA, pp: 156-160.
- [16] **Rist R**, 1996, Teaching Eiffel as a first language, *9th Journal of Object-Oriented Programming - Issue One*, *SIGS Publications Inc*, New York USA, pp: 30-41.
- [17] **Roberts E**, 2001, An Overview of Mini Java, *Proceedings of the 32nd SIGCSE technical symposium on Computer Science Education*, Volume 33 Issue 1, *ACM Press*, New York - USA, pp: 1-5.
- [18] **Robins A, Rountree J, and Rountree N**, 2003, Learning and teaching programming: A review and discussion. *Computer Science Education - Volume 13 Issue 2*, *Routledge*, Oxford England, pp: 137-172.
- [19] **Scott A, Eyres D and Watkins M**, 2006, Animated Flowcharts as an Aid to Learning Programming, *Proceedings of the 10th Java in the Internet Curriculum Conference*, *The Higher Education Academy*, London Metropolitan University –UK, pp: 12-16
- [20] **Smith P and Webb G**, 1998, An Overview of a low-level Program Visualisation Tool for Novice C Programmers, *In Proceedings of the Sixth International Conference on Computers in Education (ICCE '98) - Volume 2*, *Springer-Verlag*, Beijing, China, pp: 213-216.
- [21] **E Soloway and J Spohrer**, 1989, *Studying the Novice Programmer*, *Lawrence Erlbaum Associates*, Hillsdale, New Jersey - USA, ISBN 0-8058-0003-4.
- [22] **Thomas L, Radcliffe M, Woodbury J and Jarman E**, 2002, Learning Styles and Performance in the Introductory Programming Sequence, *Proceedings of the 33rd SIGCSE technical symposium on Computer Science Education*, Volume 34, Issue 1, *ACM Press*, Cincinnati -Kentucky - USA, pp: 33-37.
- [23] **Tudoreanu M**, 2003, Designing effective program visualization tools for reducing user's cognitive effort. *In Proceedings of the 2003 ACM Symposium on Software Visualization*, 2003, *ACM Press*, San Diego – California – USA, pp: 105-213.
- [24] **Vygotsky L, Rieber R, Carton A and Minick N**, 1987, *The Collected Works of L. S. Vygotsky. -Volume 1*, *Plenum New York USA*, ISBN:030642441X
- [25] **Watts T**, 2004, The SFC editor a graphical tool for algorithm development, *Journal of Computing Sciences in Colleges - Volume 20 – Issue 2*, *Consortium for Computing Sciences in Colleges*, Spartanburg – South Carolina - USA, pp: 73-85
- [26] **Westphal B, Harris F and Fadali M**, 2003, Graphical Programming: A Vehicle for Teaching Computer Problem Solving, *33rd ASEE/IEEE Frontiers in Education Conference*, *IEEE*, Boulder Colorado, pp: 19-23
- [27] **Winslow L**, 1996, Programming Pedagogy -A Psychological Overview. *SIGCSE Bulletin – Volume 28 Issue 3*, *ACM Press*, New York USA, ,17-22.
- [28] **Wood D, Bruner J, & Ross G**, 1976, The role of tutoring in problem solving, *17th Journal of Child Psychology and Psychiatry*, *Blackwell*, Oxford – UK, pp: 89-100.