

The Learning Psychology of Visual Programming for Object-Orientation

Mark Ireland, Roger Stone and Ray Dawson
Department of Computer Studies
Loughborough University of Technology
Loughborough, Leicestershire,
LE11 3TU, UK
Email: M.Ireland@lut.ac.uk

Keywords: POP-I.B training design, POP-II.A novice, POP-III.C visual languages, POP-IV.A object oriented design

Abstract

A teaching environment featuring a visual object-oriented programming language is an improvement over existing object-oriented teaching methods using textual languages. We propose to create such an environment, which will provide a complete course in object-orientation. The course will be based on the key object-oriented concepts and mechanisms. The visual programming environment will avoid the problems other environments have when used for teaching, as it will not feature explicit meta-classes and metaphors and will be kept simple to avoid confusion. The visual programming tool will be object-centred and object-focused, and feature a direct manipulation interface, to reinforce object thinking. It will be flexible to allow staged learning development with intermediate tests.

Introduction

This paper presents a proposal for a course in object-orientation using visual programming. For the course to be successful it must be designed with student learning psychology in mind to better enable the students to fully grasp the fundamental concepts involved. The purpose of this paper is to elicit some feedback on our proposal to provide a medium for enhancing the learning process of students who will take the course.

Object-orientation is widely acknowledged as an area of computing which is difficult to learn. The principles themselves may not be very difficult to grasp but the deep understanding of the concepts needed to produce effective object-oriented solutions to problems is hard to achieve. The proposal outlined in this paper is to provide a computer-based learning environment for object-orientation, with the intention of both making it easier to learn and instilling a greater understanding in the students.

We are proposing that the hub of the environment will be a visual programming tool of our own devising. The main feature of the tool and emphasis of this approach is direct, graphical manipulation of objects and their relationships to create programs. We believe that combining the two research areas of object-orientation and visual programming will be successful, as both areas are claimed to take advantage of the way that people think. It is claimed in the former that people think more effectively in terms of objects and in the latter that mental visualisation aids understanding of difficult concepts. It is envisaged that the use of a visual

programming environment will stimulate and help the students to mentally visualise objects and their behaviour and consequently understand the underlying concepts.

The course will consist of a set of lessons available through the computer. A fundamental part of those lessons will be interaction with the visual programming tool. A complete course in all aspects of object-oriented analysis, design and programming will eventually be provided with the first stage of its development covering the programming part of the course.

Teaching Object-Orientation

Our approach to teaching object-oriented programming is to focus on the key concepts. The course will focus on each concept in turn, isolating it, and putting it in context with previously learnt concepts. It will be emphasised how these concepts relate to the major factors of software engineering: reliability, extendibility, reusability and compatibility [Mey88]. The following list of concepts has been derived from much study of the sometimes contradictory literature:

- abstraction
- encapsulation
- information hiding
- classification
- persistence
- polymorphism

This concepts-based approach differs from that of many existing object-oriented courses which, through being based on textual object-oriented programming languages, by necessity are largely concerned with the mechanisms employed by these languages. In order to put the knowledge of concepts to use, it is necessary to introduce, where needed, the object-oriented mechanisms. We have produced the following list of fundamental mechanisms:

- Object
- Attribute
- Method
- Message
- Class or Prototype
- Inheritance

To put these lists into perspective, the inter-relationships of the software engineering factors and the object-oriented concepts and mechanisms can be considered. The software engineering factors have been formulated so that good software can be produced economically. The set of object-oriented concepts is one way of modelling software to achieve those factors. The mechanisms are the tools which allow a programmer to program using the object-oriented concepts.

Many courses on object-orientation are based on a specific object-oriented programming language, or start with a language so that the course can have a practical element before it is finished. This objective is very important, as a course needs a good balance of theoretical and practical work. Unfortunately, there are problems with using a textual language. Much time is spent learning the language and concentrating on the low-level issue of language syntax. The key concepts which need to be emphasised are essentially high-level ideas and

need to be evident from the start. In textual languages they are hidden behind the syntax and their importance is not evident. By the use of a visual programming language, we intend that the students will be programming directly with objects. The main graphical constructs they manipulate will be representative of high level mechanisms e.g. objects or methods. The visual syntax will be kept simple, and also directly illustrative of the mechanisms. After the course the students will migrate to using Smalltalk.

Smalltalk

A lot of present courses in object-orientation use Smalltalk e.g. the Open University object technology course [Sha94]. Smalltalk is considered by many to be the best textual object-oriented teaching language. It has the advantage over languages such as C++ of being designed solely for object-oriented programming. It does have a number of faults as a teaching language though. A widely described problem area is the class library, which is large and confusing but necessary for the creation of even the simplest of programs. The size makes it daunting for students, who do not know where to start looking at it, or how to start using it. The extensive use of inheritance in the library is confusing for students who are not yet comfortable with simpler concepts such as class and object.

The concept of metaclass has been considered the single most important source of learnability problems with Smalltalk [Osh86]. The metaclass is missing from other languages and is perhaps an over-elaboration on the part of the language designers, given their aim to ensure that all parts of the language were objects themselves. It is quite possible to hide class methods for construction, destruction and copying of objects, as well as class variables, within the syntax of an object-oriented language, and therefore to ignore the metaclasses completely. Our proposed visual programming system will have a more manageable class library and will not confuse the students with the concept of metaclass at an early stage. It will also have other benefits, outlined later in this paper.

Critical Incidents

Carroll, Rosson and Singley have used the theory of 'critical incidents' in an analysis of the problems of students learning an object-oriented language and developed a theory of 'critical threads' [Car93]. The proposed learning environment will aim to support the learning process at the critical incidents and also prevent the development of critical threads. Early trials of the learning environment are planned, which will highlight the most likely problem areas. By reducing the risk of misconceptions the students will be better able to understand the fundamentals of object-orientation.

Previous Work

Previous work in the area of visual programming has produced some systems which are worth considering for use in object training. Here we consider some of the most significant.

Rehearsal World

Finzer and Gould have produced 'Rehearsal World' [Fin84], a visual programming tool for the prototype model of object-oriented programming, designed for non-technical users but valid as an educational tool. Unfortunately, they built this tool to work through (and with access for the user to) Smalltalk, which is based on the class model of object-oriented programming. This is a possible source of major confusion for trainees. The metaphor used is of a theatre, which is excellent but hides the object-oriented concepts from the user. This makes it a good tool for those wanting to program using 'theatre' concepts --- the intention of the designers --- but not for those wanting to learn object-oriented concepts.

Moliere

The theatre metaphor has since been used by Borne in the 'Moliere' environment [Bor93]. This environment is more advanced than 'Rehearsal World', featuring a visual language for a small subset of Smalltalk. It suffers from the same faults when applied to object-oriented teaching: a prototypical model on top of a class-based model and an extra layer of metaphor.

Object Design Exploratorium

The 'Object Design Exploratorium' environment [Rob94] produced by Robertson et al. is for the teaching of object-oriented design principles. The system allows trainees to create designs, given the tutorial scenarios on offer. It then checks those designs against a model answer and gives feedback, which is no easy task for a computer program. This approach has potential, but is dependent on artificial intelligence for an implementation, which at present is not capable of performing the task well. The system does not cover programming, which would have to be taught separately.

This environment is also notable for the new approach to object-oriented learning which was developed along with it. It is based on 'discovery learning' where information is gained through working on tasks rather than listening or reading. Tasks, or exercises, are meaningful from the start of the course. The processes and practices of identifying and analysing objects are emphasised. Correspondingly, purely syntactical and notational details are de-emphasised. We are incorporating these principles into the design of our environment, but balancing them with the need for structured reading material.

The Mjolner Environment

The 'Mjolner' Environment [Hed88] is an object-centred programming environment. It was designed for industrial use but has potential for teaching purposes. It emphasises the direct representation and manipulation of objects which we also advocate.

How Our Environment Will Differ

Our environment is being designed for teaching purposes and not for industrial use, unlike most of these systems. It is not being designed for ease of programming by non-technical users, but for technical users with little experience of object-orientation. Therefore it will not

feature an extra layer of metaphor. It will cover the whole of the object-oriented software engineering approach, but the first focus will be on programming.

The Proposed Learning Environment

Computer-based learning is becoming an increasingly popular method of providing a course, particularly in computer-based subjects where usually both the students and the course-providers have no technology-barrier to overcome. Complete courses can now be constructed from hypermedia (text, graphics, sound and other media made available from the computer) and are termed *courseware*.

Our courseware will place an emphasis on student interaction with the material and direct manipulation of objects and their relationships. The tool will allow the creation and graphically-animated execution of programs. One of the major research aims of the work is to establish that such a high-level manipulation and observation of object-oriented programs leads to an easier and greater understanding. We are investigating methods of measuring the effectiveness of the visual programming tool and would particularly welcome suggestions in this respect.

The computer-based environment will consist of two main parts: the hypermedia courseware and the design and programming tools.

The Hypermedia Courseware

The hypermedia courseware will be the hub of the course in object-orientation. It will consist of a series of lessons produced as hypermedia. Associated background information and related information will also be available as hypermedia for students to reference and read further about the subject. The hypermedia will be arranged as pages. All the pages are made available through the TICSS hypermedia authoring system.

The TICSS system is the standard hypermedia course provision system used in the Department of Computer Studies. It constrains courses to a recognisable format and provides a number of useful features, such as searching and submission of work.

Password protection of courses is a feature of the TICSS system, but protection of individual lessons is not. Therefore, it is possible for a student to progress from one lesson to the next without having fully understood the necessary concepts of the earlier lesson. This has the advantage of allowing the student to browse ahead and get a feel for the content of the course as a whole. On the other hand, the course particularly aims to enable students to gain a good understanding of object-oriented concepts, with the belief that each concept is best learnt in turn. To further this aim, there is password protection of the features of the design and programming tools.

To determine whether a student has understood the concepts within a lesson and how those concepts relate to previously learnt concepts, two methods will be used. One method is multiple-choice tests which will be conducted through the courseware and assessed

automatically. The other method is production of programs, classes and program designs with the design and programming tools, which will be assessed by the tutor. If a student is found to have not fully understood the necessary concepts, the courseware or the tutor, respectively, will be able to give an indication of which material should be reviewed. In addition, exercises could be redone or new exercises set.

To illustrate the ideas within a lesson, the courseware will remotely control both the design and programming tools. Thus demonstrations of these tools will be used as examples of object-oriented concepts, mechanisms and techniques in use. The courseware will also remotely configure the tools for the student to explore an idea or to start an exercise.

The Design and Programming Tools

It is envisaged that there will be separate design and programming tools, and possibly an analysis tool too.

The design tool will support the production of classes, which will be arranged in hierarchies. The primary design focus has been on the programming tool. The programming tool will be a single visual programming tool dealing with the construction and execution of programs. It will allow an exploratory approach to learning object-oriented programming and the concepts behind it. The objects themselves and the interactions between them will be the focus of the tool with the intent to allow easier grasping of the fundamental concepts of object-orientation.

Object-Centred Environment

The tool will be object-centred rather than view-centred. The classification of object-oriented programming tools in this manner is widely documented [Gru95,Cha95z,Hed88]. The two categories are also known as object-oriented and action-oriented, respectively.

A view-centred tool provides multiple visual and textual views to support visualisation and construction of objects. Mappings between these views may be only partial, but are consistent [Gru95].

An object-centred tool provides concrete notions of objects rather than tools which abstract them. Each object has identity, must never appear more than once on screen and must always appear in the same, malleable, form [Cha95z]. The advantage of this form of environment for the learning of object-oriented concepts is that the student can work directly with the objects and not abstracted views of them, therefore it strongly reinforces the object concept. To view or modify properties of an object, the student must access them through that object. This makes it easy for the student to accept that the representations of objects being used within the tool are the objects which make up the program.

In comparison, a view-centred tool fragments the objects e.g. one tool is needed to see all references to an object and another to show its instance variables. It also distances the object as tools are needed to access it. It has been found that with frequent use though, a programmer can identify tools with the objects i.e. the tools become transparent [Cha95z]. The prospective users of our tool are students however; they need to have an interface which can be used for a short period with a minimum of effort, which means the best approach is object-centred.

Object-Focused Viewpoint

A strong feature of the tool will be to simulate program execution from the point of view of any object. This gives the opportunity to see directly the activity of an object, so it can be determined if it is behaving correctly. A step-by-step execution facility will allow the changes in state of an object to be followed as the program executes, showing clearly the effect of messages arriving through graphical animation.

The student will choose to have one particular object, or to always have the currently active object as the focus of execution. The objects can be arranged on the window space as the student sees fit.

Adaptation to Stage of the Course

An important feature of the environment will be the adaptation of the tool to each stage of the course. The same tool will be used throughout but at the start of the course it will provide only simple features. As the course progresses new mechanisms will be introduced. For example, at the start the tool will be based around independent objects without any notion of class or inheritance as these will not be introduced until later. The student will access the various stages of the tool through the use of a password, which will be given on successful completion of a lesson of the courseware.

In its most advanced stage, the tool will provide the whole set of mechanisms detailed earlier in this paper. The features will be introduced in a 3-step process. The first stage is to give an object-based approach, which will involve the object, attribute, method and message mechanisms. The second stage will be class-based, introducing the class mechanism and the final stage will add inheritance.

The object-based stage will illustrate, from the start, the need for a useful library of objects. It will actually be a class library, but the concept of class will not exist at this stage. The student will be unable to create new classes but will be asked to construct programs from the library of objects available.

Current State of Development

At the time of writing the environment is still in the design stage, and constantly changing. The current state of the work can be seen on the World Wide Web, at:

URL:<<http://hpc.lut.ac.uk/~comil/work.html>>

The design will also be presented at the Workshop.

Conclusion

Visual programming methods complement object-oriented methods by stimulating natural ways of thinking. A teaching environment featuring a visual object-oriented programming language is an improvement over existing object-oriented teaching methods using textual

languages. We propose to create such an environment, which will provide a complete course in object-orientation.

The key object-oriented concepts and mechanisms have been identified for teaching purposes. Early tests will be made to discover and verify the points at which students become confused.

The visual programming environment will not have the problems other environments have when used for teaching. Specifically it will avoid problems with metaclasses and metaphors and will be kept simple to avoid confusion.

The environment will use Loughborough University's TICSS hypermedia authoring and course provision system to provide course notes and exercises. The visual programming tool will be object-centred and object-focused, and feature a direct manipulation interface, to reinforce object thinking. It will be flexible to allow staged learning development with intermediate tests.

Bibliography

- [Car93] John M. Carroll and Mary Beth Rosson and Mark K. Singley, *The Collaboration Thread: A formative evaluation of object-oriented education* Empirical Studies of Programmers: Fifth Workshop, 1993, pp. 26-41.
- [Mey88] Bertrand Meyer, *Object-Oriented Software Construction*, Prentice-Hall, 1988.
- [Mey93a] Bertrand Meyer, *Toward an Object-Oriented Curriculum*, Journal of Object-Oriented Programming, vol. 6, no. 2, May 1993, pages 76-81
- [Sha94] H. C. Sharp and M. A. Newton, *Object-Oriented Analysis and Design: A Question of Approach*, in: Proceedings of the First International Conference on Software Engineering in Higher Education (SEHE 94), 1994, pp. 309-316.
- [Osh86] Tim O'Shea, *Panel: The Learnability of Object-Oriented Programming Systems*, in: Proceedings of (OOPSLA) 1986. ACM SIGPLAN, pp. 502-504
- [Fin84] W. Finzer and C. Gould, *Programming by Rehearsal*, Byte, vol. 9, no. 6, pages 187-210, June 1984
- [Bor93] Isabelle Borne, *A Visual Programming Environment for Smalltalk*, Proceedings of the 1993 IEEE Symposium on Visual Languages, 1993, pp. 214-218.
- [Rob94] Scott P. Robertson and others, *ODE: A Self-Guide4 Scenario-Based Learning Environment for Object-Oriented Design Principles*, ACM SIGPLAN Notices, vol. 29, no. 10, October 1994, pp. 51-64
- [Hed88] Gorel Hedin and Boris Magnusson, *The Mjolner Environment: Direct Interaction with Abstractions*, in: Proceedings of ECOOP '88, European Conference on Object-Oriented Programming, pp. 41-54, 1988.
- [Gru95] John Grundy and others, *Connecting the Pieces*, in: Visual Object-Oriented Programming Concepts and Environments, eds.: Margaret M. Burnett, Adele Goldberg and Ted Lewis, Manning Publications, 1995
- [Cha95z] Bay-Wei Chang and David Ungar and Randall B. Smith, *Getting Close to Objects*, in: Visual Object-Oriented Programming Concepts and Environments, eds.: Margaret M. Burnett, Adele Goldberg and Ted Lewis, Manning Publications, 1995