

VISUAL MATERIAL AND LEARNING AIDS IN TEACHING OBJECT ORIENTED PROGRAMMING

Dr Hong Guo
Department of Computer and Network Systems
Faculty of Engineering and Computing
Coventry University,
Coventry, West Midlands, UK
h.guo@coventry.ac.uk

ABSTRACT

Based on several years of observation, investigation and analysis of students' learning behaviours we believe that identification and adoption of innovative ways of teaching computer programming in higher education is crucial. A dynamically delivered visualization-based approach was developed with the intention both of making programming concepts easier to understand and of improving the students' mental model of how a program is "working".

To help students develop a disciplined and systematic approach to their learning and to encourage an effective, methodical way of thinking to the solving of programming problems, learning aids were designed and employed in our teaching.

A survey was conducted to verify the effect of using dynamic visual teaching materials and the learning aids. An increased pass rate was achieved and positive comments were received, which indicates a preliminary success.

Keywords

Object oriented programming (OOP), Java, Visualization, Learning aids.

1. INTRODUCTION

Teaching programming is regarded as problematic. From our experiences, the area of teaching object-oriented programming to undergraduate students is one that has produced increasing problems with regard to high failure rates and poorly developed programming skills. As described novice programmers often have great difficulty in understanding what the computer is actually doing when it executes a program [1].

Effective learning of object oriented programming comprises two essential elements. One is the comprehension of many theoretical, complex and

highly inter-related concepts, which are usually grasped from lectures and books. The other is the engagement in practical activities which transform this theoretical knowledge into useful programming skills. These skills have to be utilised in the creation of solutions to problems that are executable on computers. A challenge facing the subject lecturers is therefore to provide an appropriate combination of theory and practice.

The traditional method in teaching in higher education is to lecture using textual descriptions and examples using static media, such as lecture notes presented via OHPs, supported by directed self-learning from reading textbooks. This approach is said to be ineffective in teaching programming [2, 3]. Simply because of their static nature these techniques are not capable of expressing the dynamic aspects of computer programming concepts and techniques. To demonstrate these concepts adequately, visual learning technology is introduced [4, 5]. An explicit visualization of classes, objects, and methods have proved to be important. However as Naps et al. stated [6] "... visualization technology, no matter how well it is designed, is of little educational value unless it engages learners in an active learning activity".

In this paper, we introduce a dynamic, visualization-based approach to teaching and a systematic way of learning to program. The emphasis is to ease the introduction of basic object oriented programming concepts and principles, and to engage students in active learning achieved by thinking actively, leading to an effective acquisition of coding proficiency.

The work is based on a pilot study carried out in 2004 – 2005 for a year 2 module which follows-on from an introductory OO programming module for students on BIT and 2+2 Vocational Computing Programmes. A diagnostic survey was conducted at the beginning of the module with the purpose of determining the level of students' understanding of the subject and identifying the most common stumbling blocks experienced during their previous learning. The results of this survey were used to decide the best way to organize and teach the module.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

© 2006 Higher Education Academy

Subject Centre for Information and Computer Sciences

2. SURVEY RESULTS AND PROBLEM IDENTIFICATION

Although much research has revealed problems in teaching introductory programming, little work has been done to discover the reasons that cause the most difficulty for students to continue learning object oriented programming. The lack of such studies may result in lecturers of programming simply relying on experience and intuition.

Drawn on several years experience in teaching the subject, a survey was carried out at the start of the module to receive views from students on the common problems they encountered during year 1 study. It was hoped that the issues could be addressed in this more advanced module. The questionnaire was split into two sections:

The first part collected general information about the respondents, e.g. what programming language they had experience with, what IDE they had used, why they choose to take this module, and their level of interests in the subject. Table 1 shows the details.

Table 1. Motivation of students in learning Java

IDE used (tick or specify)	Reason to come on this module (tick or specify)	Describe the level of your interests or feelings to Java programming (tick more than one, specify why)
BlueJ jGrasp	a. compulsory b. free choice c. because of Java is important for career d. to improve programming skills	a. hate b. boring c. too hard d. challenge e. useful to learn f. some interest g. easy & fun h. enjoyable

The responses of the 42 students showed that 58% did not like the subject even though some of them (13%) were taking it as a free choice. 87% were compelled to take it. All students had passed an introductory OOP module in year 1. After their introductory module, 46% of students found that Java programming is most difficult to learn. Although the majority of students (56%) had a positive motivation to learning the subject, with the view that programming skills are necessary for a lucrative career, some responded that they hated it.

The second part formed the bulk of the questionnaire, presenting a series of questions that aimed to show and explore their general knowledge of the basic but essential underpinning subject concepts and coding ability that this module is supposed to take as its starting point. Programming is a subject that builds continuously. If a student fails to grasp the fundamental concepts or principles, it can lead to increasing difficulty in coping with the pace of the teaching, which is essentially driven by the curriculum, rather than the learning of the student.

Brief results are presented in Table 2. The responses revealed that over 80% of students failed to provide correct answers to the questions related to the key concepts in object-oriented programming. Most students could not spot the most common syntactic errors. 80% of them professed a complete inability to write even the simplest program. Without reference material, they simply cannot write any code "off the top of their head"; a poor memory for programming syntax is evident.

Table 2. Results of the subject questions

Questions	Percentage of right answer	Percentage of wrong answer
In Object-Oriented programming, what is the difference between a class and an object?	15%	85%
In addition to a body of a method, there are three other parts to a method definition. Name each of these parts.	19%	81%
What general name is given to the method(s) that initialize an object's data members when an object is instantiated?	6%	94%
The code for class Box has 7 errors in it. Spot and correct it.	11% found three errors, 56% found four errors, 22% found five errors, 6% found six errors.	
Assuming the above errors have been fixed and you need to test the Box class. Write a simple class TestBox containing only the main method, which will create one instance of the Box class and output the height, width, depth and volume of the instance.	21% attempted only a few were correct	79%
Add a method to the class Box which calculates the surface area of a box. Modify TestBox so as to output the surface area, and the ratio of volume to surface area for the instance of the Box class	9%	81%

By our observation, students tend to spend little time in careful thinking during the analysis and design stages when attempting tasks or problems set at an early stage of a module, which are simple and easy to solve. Common behaviour is to quickly write code line-by-line without having any bigger program structures in mind, and when trying to correct their errors use small local fixes without making mental notes about their reasoning. When they are lucky their program compiles and executes. Even then they unfortunately do not always have a clue as to whether the output is correct, as they had never thought about what they would expect of the

program. They often get very frustrated when attempting more complex tasks, such as an assignment, for two main reasons: 1) they become confused and don't have a clue as to how to attempt it; 2) they have no way of knowing where to look to correct the long list of error messages. More damagingly, many students may not even be bothered to attempt any of the practice exercises set simply because such efforts do not count for marks. There is generally no indication of potential failure during the course itself, as there is no way of checking and monitoring students' performance until all assessments have taken place, by which stage there is little we can do to improve the situation.

Combined with many years of investigation and analysis of students' learning behaviour, attempting to identify the major difficulties and the root causes of students' learning difficulties, a conclusion can thus be drawn. This is that the inability to grasp the fundamental OO concepts from an early stage and the lack of experience in practical program coding together play a prominent role in undermining students' confidence and their ability to succeed.

It is therefore desirable to adopt an innovative teaching approach which enhances student understanding of the basic OO concepts, supported with a sufficiently transparent learning process that enforces a systematic development of thinking in the solving of programming problems. Learning aids were thus devised and used to assist this purpose. The learning aids include a logbook, clearly defined guidelines, and well-specified self-assessment criteria. The following sections discuss their use.

3. IMPROVEMENT IN TEACHING OOP

Students who are learning Java programming frequently have a difficult time understanding the paradigm and how it operates. They cannot see the relationships between objects: that exist physically in reality, that are described conceptually in scenarios and books, that are written in computer programs, and that are constructed and used inside the computer's memory during the program's execution. This means that they find it hard to understand the relationship between their code and what was happening when the written program executes.

Visual learning proves to be a great help which benefits students in gaining a clearer understanding of the programming principles. It effectively provides students with an insight into the computer system. In this paper, we describe a dynamic, visualization-based approach to teaching, which was designed to make the subject knowledge easier for the student to understand and to improve the students' mental model of how a program is "working". Our lectures aim to introduce the basic object oriented concepts from discussions of real-world, tangible examples, and then move to using visual materials to support the learning of a programming language.

Visualization is achieved by using animated PowerPoint slides which include dynamically displayed pictures, graphical diagrams, and line-by-line explanations of what is happening in memory at each stage in the execution of a program fragment. Lectures, using these clearly presented and easy to follow slides, will engage students in active thinking which is often driven by questions from instructors. Teaching in this interactive way will motivate students' participation in classes and the use of dynamic means showing step-by-step changes in both computer memory and in objects' state will benefit students in creating a clearer mental model of program execution. (Due to limitation in space, a detailed discussion of the approach adopted will be described in other paper which is in writing.)

However, programming is a subject where students must practice regularly in order to become proficient. Regardless of how effective attending lectures or reading books are in developing an understanding of the subject's principles, without substantial efforts in programming activities and adequate experience to allow students to visualize by observing the effects of program execution, there is in our view, little chance of most students developing effective programming abilities.

Biggs [7] emphasizes the importance of activities in which the student engages as part of their learning, and indicate that the teacher's role is not only to devise suitable tasks, but also to make sure that the students do indeed engage in them. To this end, a set of carefully designed tasks and exercises is issued each week as a worksheet which students are encouraged to undertake and complete within that week, in order to consolidate the theoretical concepts taught that week. Students are required to log their programming activities in their logbook.

3.1 Effect of using a Logbook

The use of logbook as a part of learning aids follows the best practice technique introduced by the domain expert, Watts Humphrey of the Software Engineering Institute, in his seminal books *Personal Software Process* [8] and *"A Discipline for Software Engineering"* [9]. The emphasis is for students to acquire programming skills through a disciplined programming process from recording their programming activities in a systematic way. Students are provided written logbook guidelines which specify clearly how the logbook should be maintained and used. It is hoped that the logbook will aid students, at a minimum, to: 1) think carefully before coding while analyzing problems and designing solutions; 2) allow them to be aware of their personal strengths and weaknesses; 3) help them to track their progress based on actual evidence; 4) reflect and improve from such a transparent learning process; 5) assist them to learn effectively from the process of documenting their own mistakes; 6) encourage students to follow and

become familiar with the programming process in a progressive and systematic fashion.

The logbook also provides instructors with a vehicle that not only allows them to monitor the students' performance by informally checking them, but also facilitates timely identification and prompt resolution of problems in the teaching and learning.

The logbook forms a substantial portion (40%) of the coursework component assessment for the module, which aims to encourage students' engagement in doing regular practice. It is also hoped that the use of a logbook will improve self-organization skills, and clearly specified guidelines can make things easier to follow hence motivate participation.

3.2 Logbook Assessment Criteria

As an element of the learning aids, explicitly defined logbook assessment criteria are handed out to students at start of the module. These criteria permit students to gain a better understanding of how they can maximize their profile. The primary temptation is to reward those students who are making satisfactory progresses in regular practice, and to highlight the importance of performing programming activities in a systematic manner.

The assessment criteria emphasize on a systematic programming process through recording all the activities that demonstrate analytical thinking. The grading is based on output evidence produced from the most important programming activities, such as, 1) analysis and design, to provide hand-drawn UML class diagrams, hand-written formulae, algorithms and pseudo-code as appropriate; 2) create a test plan, which makes clear the test data and the expected output, including hand-drawn UML object diagrams; 3) coding and compiling, to transform the class diagrams into code using an IDE. Hand-written code is not acceptable in this case. Students are required to record any new error messages that they encounter along with a brief description of the cause and the correction made; 4) testing, to write test classes according to their test plan. Students are required to stick the printouts of working-code listing and the actual output of execution into logbook for evidence.

3.3 Self-assessment

Use of self-assessment supported by explicitly defined criteria leads to a better knowledge of what is required and what can be done to improve marks.

The logbook is assessed in three phases giving a total of 40% of the coursework component mark. The first phase is worth only 5% in the form of self-assessment. Students need to complete and hand in the self-assessment form provided. The instructor checks and signs in the logbooks. This is conducted at week 3 and acts as an alert for those who have not yet got a logbook to do so.

The second phase is worth 15%, conducted at the halfway stage in week 9, which aims to provide timely feedback on any common problems and to identify potential failures, in which case warning comments will be given in the logbooks. The logbooks are handed in with a completed self-assessment form for reference to the instructor, and are then marked and returned to students before the Christmas vacation to allow catching up over the break if they wish.

Towards end of the module students hand in their logbook for the phase 3 assessments, worth 20%. This is marked for overall performance. It aims to allow more opportunities to those who did not do well in their first two phase assessments to improve and to encourage continue improvement.

Student participation in self-marking could make the overall marking process more transparent and consequently give students a better perception of their individual performance compared to the cohort.

4. EVALUATION

A questionnaire-based evaluation was conducted to investigate the effect of various module components and student's views on the main features of the module. Table 3 shows the result from 42 students.

Table 3. Results of the survey

How useful are	Very useful	Useful	Not very useful	Useless
Visual materials in lectures?	18%	64%	18%	0%
Workshops?	5%	23%	59%	14%
Recommended textbook?	5%	73%	23%	0%
Worksheets exercises?	18%	77%	0%	5%
Logbook as a learning aid to problem solving skills?	5%	68%	18%	9%
- Logbook guidelines?	14%	45%	36%	5%
- Logbook self-assessment form?	9%	36%	45%	9%
- Logbook marking criteria?	9%	41%	36%	14%
Solutions of the worksheets?	41%	55%	0%	5%
Materials on WebCT, e.g. code online?	9%	45%	27%	18%
Test in class?	5%	59%	36%	0%
Logbook assessed in three phases?	0%	73%	18%	9%
Assignment?	23%	50%	27%	0%

The statistics of the module pass rates were collected for the last three years of the module delivery, one using learning aids plus visual materials in teaching, compared with the previous

two years taught without them. Table 4 shows the improvement.

Table 4. Increase of pass rate on the course

Java module	Percentage of pass rate
2002 - 2003	42.4%
2003 - 2004	44.7%
2004 - 2005	62.2%

5. DISCUSSION

Among many problems we have identified that students seem to experience in learning object-oriented programming, four are most troublesome: 1) acquiring a firm understanding of the basic, but interrelated concepts of classes, objects and the essence of “objects communicate by sending messages”; 2) the ability to visualize the effects of program execution, with respect to changes in computer memory, and changes in the objects’ state; 3) adopting a disciplined and systematic process to the creation of computer programs, towards an effective development of problem-solving skills; 4) the ability to understand and sort out errors messages during compilation, especially when working on slightly larger and more complex programs, such as an assignment.

The effect of using a visualization-based approach to teaching OOP is significant. Students can become more motivated if lectures are easy to follow and comprehend. Visual materials in lectures can play an effective role in making the OO concepts much more readily understandable.

Object oriented programming is not learned by reading, but by practicing. The use of learning aids, consisting of a logbook, well-specified logbook guidelines and explicitly defined assessment criteria, has demonstrated that they can have an important role in the effective learning of how to program, and has improved students’ confidence in dealing with reoccurring problems in their programming practice. The students’ reflective feedback has acknowledged the benefit of using logbook.

The results of the evaluation confirms that the use of learning aids motivated students' participation and yielded a pass rate increase of 17% as shown in Table 4.

6. CONCLUSION

This paper has identified that students being unable to grasp the fundamental OO concepts from an early stage and lacking experience from practice play a prominent role in undermining novices’ confidence and their ability to succeed.

A dynamically delivered visualization-based approach to teaching has been used with the intention both of making programming concepts easier to understand and of improving the students’ mental model of how a program is “working”.

Learning aids that aimed to help students develop a disciplined and systematic approach to their programming process, leading to a methodical way of thinking in the solving of programming problems, has been designed and employed in our teaching. The benefit gained from the introduction of the learning aids was significant.

The results of the module survey revealed that the use of dynamically presented visual materials in lectures and the learning aids have been effective.

An increased pass rate was achieved and some positive comments received which suggests that their introduction was a success.

Unfortunately due to limitation of paper length, it is not possible to describe the visual materials in any detail here. Examples of such will be presented at the conference.

7. ACKNOWLEDGEMENTS

The author would like to thank Lisa Payne, who contributed valuable comments and many helpful ideas to this project.

8. REFERENCES

- [1] Rowe, G., Thorburn, G., (2000), VINCE – An On-Line Tutorial Tool for Teaching Introductory Programming. *British Journal of Educational Technology*, 31 (4), pp.359-369.
- [2] Bligh, D.A., (2000) *What's the Use of Lectures?* Jossey Bass Wiley, ISBN: 0787951625
- [3] McDonald, C. and Patterson, A., (2003) *Why Lecture? The 4th Annual Conference of the LTSN Centre for Information and Computer Sciences*, Galway.
- [4] Dershem, H.J., Vanderhyde, J., (1998) *Java Class Visualization for Teaching Object-Oriented Concepts*, Proceedings of the twenty-ninth SIGCSE technical symposium on Computer Science, Atlanta, Georgia, United States 1998 ISSN:0097-8418.
- [5] Allison, I., Orton, P. and Powell, H., (2002) *A virtual learning environment for introductory programming*, Proceedings of the 3rd Annual Conference of the LTSN-ICS, August 2002
- [6] Naps, T. L., Roling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Velazquez-Iturbide, J. A. (2003) *Exploring the Role of Visualization and Engagement in Computer Science Education*. *ACM SIGCSE Bulletin* 35(2), pp.131-152.

- [7] Biggs, J., (1999) Teaching for Quality Learning at University. OUP / SRHE
- [8] Humphrey, W.S., (1997) Introduction to the Personal Software Process, Addison-Wesley, ISBN 0-201-54809-7.
- [9] Humphrey, W.S., (1995) A Discipline for Software Engineering, Addison-Wesley, ISBN 0-201-54610-8