Student Perceptions of Learning Object-Oriented Programming

Leila Goosen
Vreda Pieterse

Abstract
This study was undertaken in order to determine how difficult (or not) CS1 students perceive learning to program in an object-oriented style to be, how well they actually learn object-oriented programming (OOP) and how well they retain their understanding of OOP. By analysing this information and ideas of best practices provided in related literature, suggestions are formulated for improving instruction of OO concepts.

Introduction

Students' acquisition of basic object-oriented (OO) concepts appears to be a major source of difficulty (Doube, 1996), because understanding these concepts requires them to be familiar with some potentially troublesome terminology (Ross, 1996). Knowing which OO concepts students find difficult to understand can allow lecturers to structure instruction in such a way that understanding is improved (Wiedenbeck & Ramalingam, 1999).

In the next section of this paper the research design will be elaborated upon by describing the problem statement, population and sampling, as well as how data was collected. The following section provide an elaboration on the theory of OO concepts, including objects and classes, state and behaviour, encapsulation and interfaces, inheritance and polymorphism.

Our research results are reported in three sections: The influence of prior learning and experience for learning to program using OO concepts; the understanding of OOP; and the identification of easy and hard topics, both generally and specifically with regard to OO topics. Understanding is interpreted in terms of the perceived and actual difficulty for students of understanding OOP, how well students retain their understanding of OO concepts, and the consistency of the latter with students' course marks.

In the last section we conclude with specific teaching guidelines for improving instruction of OO concepts.

Research Design

Problem Statement

The perception seems to exist that students have difficulty in learning to program in an object-oriented style (cf. SIGCSE, 2001). This study was undertaken in order to determine how difficult (or not) students perceive learning to program in an object-oriented style to be, how well they actually learn object-oriented programming (OOP) and how well they retain their understanding of OOP.

Population and Sampling

Our study was conducted using students who completed an OOP course at the University of Pretoria (UP) during the first semester of 2002. The following table shows the courses presented by the Department of Computer Science of UP and their prerequisites as it was presented in 2002 when this survey was conducted.

Table 1: First year programming courses at UP

Course Code	Content	Prerequisite
COS 110	Programming design principles and practices with emphasis on the object-oriented paradigm using Java as vehicle	Grade 12 Computer Studies (HG) D or an equivalent
COS 283	The course is an introduction to networking principles using Java for www and network programming.	COS110

Students who participated in our survey were enrolled for COS283 at the time of our survey, and therefore had completed COS110. It can thus be assumed that they are schooled in the concepts of object-orientation. Due to the prerequisites set for COS110, all students had programming experience in at least one programming language prior to their exposure to OOP in COS110. It should be noted that 25 students (12%) had not yet passed COS 110 at the time of the survey, but had obtained special permission to do COS 283 in parallel while redoing COS 110.

Data Collection

On October 7, 2002 all students attending class in COS 283 were invited to participate in a questionnaire. The attendance register of that day had 246 students, of whom 178 (72%) completed the questionnaire. As the questionnaire was completed anonymously, the final marks for COS 110 of all attendees of the day were obtained, together with the period (semester and year) when each student had completed the course. Of these students, 197 (80%) had passed COS 110 in June 2002, meaning the majority of students completed the questionnaire roughly four months after completing a course teaching them Object Oriented Programming.

Object Orientation Concepts

For the purpose of this research, understanding of the concepts of object-oriented programming described below is considered to be crucial in the mastering of OOP.

Objects and Classes

An object can represent any abstract or real world entity (Finch, 1998) that is relevant to the system, while "classes are a means for describing the properties and capabilities of the objects in real life that a program has to deal with" (Bishop, 1998:23).

State and Behaviour

An object's internal state is defined by the values of its attributes, which can be thought of as a collection of variables and data. The concept of attributes is central to object-oriented programming

(Barrow *et al.*, 2002:3). The behaviour of an object is defined it its methods. Methods specify the operations an object can perform. Among other things, it defines the way in which an object's data can be manipulated (Martin & Odell, 1992:17) to change its state.

Encapsulation and Interfaces

Objects are self-contained entities that encapsulate (hide) both their own data and functions (Barrow *et al.*, 2002:466) as described in its class definition. The interface of an object presents the visible surface of the object for other objects to communicate with it (Chakravarty & Lock, 1997:122). An object thus has control over the way in which other objects can access its data and methods, and can hide anything that it wants to keep private from other objects. The details of an object code belong to the class itself, and this code may be modified in any way desired, as long as its interface remains unchanged (Schneider & Gersting, 2004:391).

Inheritance

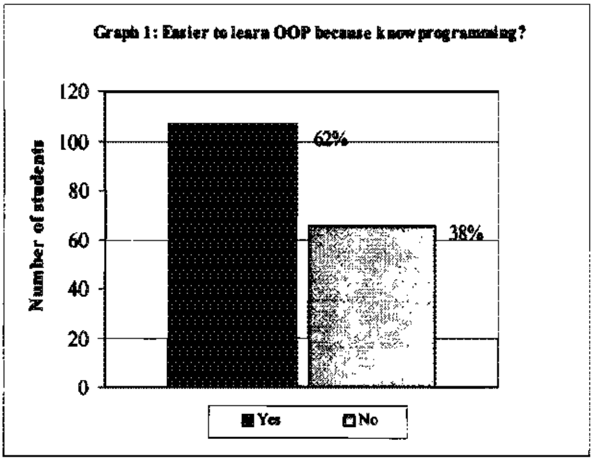
Use of OO design permits class definitions to be hierarchically arranged (Barrow *et al.*, 2002:476), leading to the powerful inheritance programming technique. Wirfs-Brock *et al.* (1990:24) define inheritance as "the ability of one class to define the behaviour and data structure of its instances as a superset of the definition of another class or classes." Programmers make use of the process of inheritance by abstracting all the common features into a high-level member class that represents the characteristics that are shared by all its descendants. Inheritance allows code and data structure definitions to be shared among classes with similar structure (Coggins, 1996), leading to the possibility of existing code being reused.

Polymorphism

The term polymorphism originates from the Greek words 'poly morph' and suggests the capacity to appear in many forms (Jupitermedia Corporation, 2003). In the context of object-oriented programming, the concept of polymorphism refers to a programming language's ability to allow different objects to react to the same stimuli (i.e. message) differently, depending on their data type or class, by redefining methods for derived classes. An object can operate interchangeably with an ancestor in relation to the attributes and methods it inherits from the ancestor (Barrow *et al.*, 2002:465).

Influence of Prior Learning and Experience

In order to establish the influence of prior learning and experience for learning to program using OO concepts, students were asked to indicate whether it was easier to learn OOP, because they already knew programming before starting this particular course. Almost of students' prior learning experiences influenced their learning of OOP positively (see graph 1).



Students were also asked to qualify their responses. Analysis of the responses of students' who indicated that their prior learning helped them, revealed that for most students, their previous programming experience and basic background knowledge of programming concepts help them mainly in a sense that there were less new concepts to deal with.

"A bicycle is a bicycle; some just have more bells and whistles than others"...This quote is from a student who is of the opinion that it was easier to learn OOP due to previous programming experience captures the feeling that the transition were more like riding a new bicycle with some new features rather than learning to ride a totally different vehicle.

Other students, despite indicating that their previous programming indeed helped them in the course under discussion, however, felt that learning the new paradigm required some unlearning of old concepts to be able to grasp OOP at first.

The words "*Even though the jump to OOP was a bit of a ...*" used by a student to describe his/her feeling, represents a general perception that they had to overcome a considerable gap between their prior experience and the new concepts.

Although the majority felt that their prior experience helped them, one student aired the opinion that the fact that he already knew programming helped him "*very little*" during this course, while another's pre-knowledge only helped "*for about the first practical lesson and then it was impossible again*".

The reasons for not gaining from prior programming experience can be attributed to the feeling that learning OOP is very different from what students were used to.

"*very different from everything ever learnt before*"

"*a whole different approach to programming*"

"*just as hard to learn as it was to learn programming originally*"

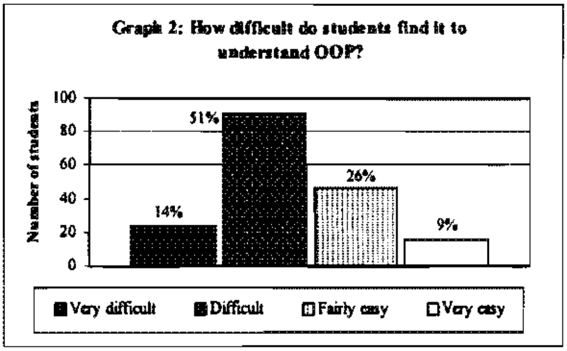
The above quotes from respondents express the frustration experienced by some students despite their prior knowledge. Their inability to utilise their previous experience can partly be attributed to inadequate depth of their prior knowledge, but is mainly seen to be a consequence of the vast difference between *structured* or *procedural programming* and *object orientation.* The following quote not only expresses the magnitude of the difference between these paradigms, but also indicates that the student has not yet adopted OOP: "*There is a big step from normal programming to OO programming*"

Many students still struggle with the ideas and find OOP to be "*still really, really difficult!*" and "*not easy to learn at all*". From these comments qualifying the students' choice whether their prior knowledge helped them to understand OOP, we conclude that although most of the students indicated that it had a positive impact, the extent of the impact was not profound in all cases. The major reason for students not being able to gain from their prior knowledge is twofold. Firstly they struggle to change their mindset to the new paradigm and secondly after obtaining the OOP mindset, they are unable to apply their previous knowledge in the new environment.

Understanding Object-Oriented Programming

Perceived Difficulty of Understanding OOP

Subjects were asked to select how difficult they perceived the understanding of object-oriented programming to be from the available options of "Very difficult", "Difficult", "Fairly easy" and "Easy". Almost 2/3 of the group indicated that they found the concept to be difficult or very difficult to understand (see graph 2).



Actual Understanding of OOP Concepts

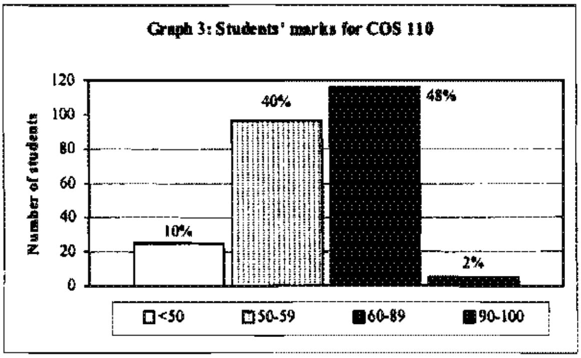
Students' formal examination marks were used as an indication of their level of understanding of the basic concepts of object-oriented programming (see graph 3). The average for students included in this study was 60 (marks out of 100) with a standard deviation of 11.

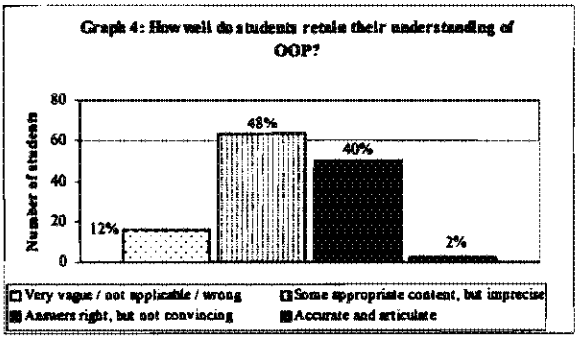
The average quoted for COS 110 might seem high. This can be attributed to the fact that COS 110 is a prerequisite for the course the students were doing when they participated in the questionnaire.

Retention of understanding of OO concepts

In order to determine students' level of retention of understanding of OO concepts, they were asked to explain in their own words what they understood by OOP. Their answers were then rated according to the following scale:

1. Very vague / not applicable / wrong.
2. Some appropriate content, but imprecise.
3. Answers are right, but don't convince the reader that the student knows exactly what the concept is about.
4. Accurate and articulate; student seem to understand the concept very well.





Classifications were made and checked by the researchers, and also verified by an external judicator. Results obtained are represented in graph 4. To illustrate the range of answers and give an indication of the way in which responses were classified, figures 1 and 2 show some excerpts.

Figure 1: Very vague / not applicable / wrong responses
One student replied to this question with a "*?*", one claims that he "*didn't understand*" and two more didn't "*know how to explain these*". It is "*difficult programming*" that is "*extremely difficult to define*", involving "*programs which are built by objects*" (2) / "*action objects*". Programming like this is "*useful*" and "*uses an interface*" for the "*look and feel of the program*" in a "*basic common language*". "*It is not mainly user-based*".

Figure 2: Accurate, articulate responses

Object-oriented programming refers to "*a design style that models real world objects as bundles of code, and furthermore describes the preferred method of interaction between these software objects*". Everything is viewed "*as an object, which has attributes and ways (functions/methods) to access and modify the attributes*", as well as information hiding. "*These objects can be treated as single entities*": "*data structures containing methods and properties can be treated as a template from which objects can be instantiated.*"

Consistency between Course Marks and Retention of Understanding

A comparison of graphs 3 and 4 show consistency for course marks and retention of understanding in terms of percentages for the extreme cases: the percentage of students who obtained marks of more than 90% in the course retained their knowledge to provide accurate and articulate responses in the questionnaire, while the percentage of students who had not yet passed the course closely matches the percentage of students who were only able to provide very vague descriptions of OOP. The inversion of percentages for the middle columns of these graphs suggest that students who had obtained marks for the course in the lower sixties had retained their understanding of OO concepts (or lack thereof) to the same extent as students with marks in the fifties.

Perceived Difficulty Level

General

The answers given by students responding to the question "*Which part(s) of Java was the easiest?*" resulted in 4% indicating that nothing was easy and roughly 74% of the respondents referring to concerns not specifically related to OO concepts. Topics mentioned ranged from use of specific data structures such as arrays or Strings or known control structures such as if statements and loops, to file handling and GUI design. Some referred to external issues regarding the quality of notes or lecture behaviour. Similar to the answers to what was considered easy, 63% of the responses to the question *"Which part(s) of Java was the hardest?"* referred to issues regarding programming in general, external factors, or topics not confined to OO programming.

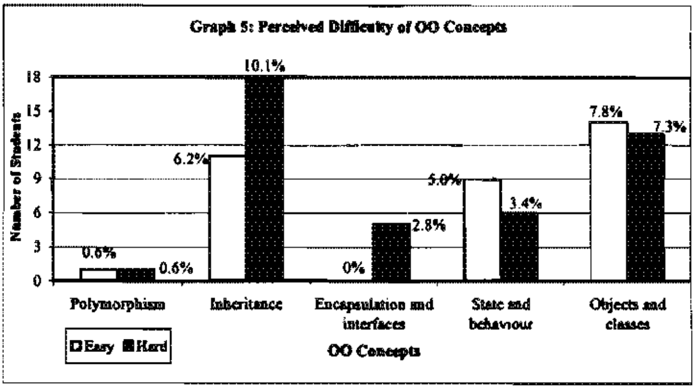
When asked to identify easy topics, 4% indicated that nothing was easy, and consequently that everything was hard. In contrast, 12% of the respondents admitted that they found everything hard when asked to identify hard topics. Similarly where only 2.8% indicated that they experienced no difficulties at all when asked about what they find hard, 7.8% were willing to state that everything was easy when asked about what they found easy. This observation illustrates that the way a question is asked can have a profound impact on the statistical result.

Specific Object Oriented Topics

20% of the respondents identified topics specific to OO that they found easy to understand, while 24% singled out specific OO concepts as being hard to understand. The number of students perceiving the identified OO concepts as being easy or hard to understand is shown in graph 5.

As this data was drawn from a small subset of the sample, no significant conclusions can be drawn from it. In the case of basic concepts such as *Objects and classes* and *state and behaviour* where more students found it specifically easy than students indicating it to be hard, it can be argued that there might be more students who found it hard, but could not even identify their inability to grasp the concept, as the origin of his difficulty in mastering the course content.

The case of *polymorphism* that was mentioned only twice (representing 1.2% of the sample) is merely an indication that the concept of polymorphism was not significantly covered or required in the presentation of the course to get an indication if the students perceived it as easy or hard. The fact that *encapsulation* and *inheritance* is perceived as harder to assimilate by students correlates with our experience that these concepts are more advanced. It is also comforting to realise that these topics was emphasised adequately enough in presentation of the course to lead the students to recognise the cognition of these concepts as crucial.



Conclusions

Our first conclusion is that although the majority of students felt that prior experience in programming helped them to understand OOP, the extent of the impact was not profound in all cases. When students are not able to gain from their prior knowledge, it is either because they struggle to change their mindsets to the new paradigm, or after obtaining the OOP mindset, are unable to apply their previous knowledge in the new environment.

The majority of test subjects claim to find understanding object-oriented programming difficult. Traditional lecture methods can be supplemented to enhance student understanding of difficult concepts by steadily replacing these by a learner-centred approach where students have more responsibility for their own learning. A pervasive idea seems to be to present concepts in broad strokes first and add details later (Ross, 1996). Students first have to be allowed to master basic facts, features and rules and gain insight into how these relate to existing knowledge (Sharp *et al.* 2003). They find it easier to understand when they familiarize themselves with high-quality, clear examples of a new OO concept, which are contrasted with non-examples. Instead of only having purely descriptive introductory material, "hands-on" programming exercises should be utilized (Doube, 1996).

This study shows a clear relationship between students' exam marks and their retention of understanding OOP concepts. As a projected outcome for students learning OO concepts is that they will eventually use these concepts in practical implementations in their place of employment, it is important that they retain their knowledge of these concepts. Exam marks should be a good indication to employers of how well a student would be able to apply OO concepts in actual projects in the work place.

Encapsulation, inheritance and polymorphism were identified as concepts within OOP with pivotal importance in terms of difficulty of understanding. As it is difficult to understand abstract concepts such as encapsulation and information hiding, it is important that the related advantages of these concepts be emphasized (Sharp *et al.*, 2003). Lecturers should also note that their attitudes help to determine whether students view inheritance as a difficult topic (Schaller *et al.*, 1997).

SIGCSE (2001) cautions that certain didactical problems can be exacerbated when an objects-based model is used, as many of the languages used for object-oriented programming in the industry involve significantly more detail complexity than classical languages. Unless lecturers take special care to introduce the material in a way that limits this complexity, such details can easily overwhelm students.

References

ACM Special Interest Group in Computer Science Education (SIGCSE) 2001. Chapter 7: Introductory Courses. *Computing Curricula 2001: Computer Science Volume.* http://www.acm.org/sigcse/cc2001/index.html [Date of access: 2 April 2003].

Barrow, J, JH Gelderblom & MG Miller 2002. *Introducing Delphi Programming: Theory through Practice.* 3rd Ed. Cape Town: Oxford University Press.

Bishop, JM (jbishop@cs.up.ac.za) 2000. Interface 2000. [E-mail to:] Southern African Computer Lecturers' Association. (SACLA@wwg3.uovs.ac.za) February 14.

Chakravarty, MM & HCR. Lock 1997. Towards the Uniform Implementation of Declarative Languages. *Computer languages* 23,2-4:121-160.

Coggins, JM 1996. Subject-Oriented Programming http://iraf.noao.edu/iraf/web/ADASS/adass_proc/adass_95/cogginsj/cogginsj.html [Date of access: 27 January 2003].

Doube, W 2000. The Impact on Student Performance of a Change of Language in Successive Introductory Computer Programming Subjects. *Proceedings of the Australasian Conference on Computing Education (ACE 2000)* (Melbourne, Australia, December 2000). New York, NY: ACM Press.

Finch, L 1998. So Much OO, So Little Reuse. http://www.ddj.com/documents/s=909/ddj9875g/9875g.htm [Date of access: 27 January 2003].

Jupitermedia Corporation. 2003. Polymorphism. http://webopedia.internet.com/TERM/P/polymorphism.html [Date of access: 23 September 2003].

Martin, J & JJ Odell, 1992. *Object-Oriented Analysis and Design.* Englewood Cliffs, N.J.: Prentice-Hall.

Ross, JM 1996. Learning to teach C++. *SIGCSE Bulletin* 28, 2: 25-30.

Schaller, NC, M Berman, J Bishop, P Nixon, E Rozanski, & P Welch 1997. Panel Report: Using Java in Computer Science Education. *The supplemental Proceedings of the Conference on Integrating Technology into Computer Science Education: Working Group Reports and Supplemental Proceedings (ITiCSE '97)* (Uppsala, Sweden, June 1-5 1997). New York, NY: ACM Press.

Sharp, H, ML Manns, & J Eckstein, 2003. Evolving Pedagogical Patterns: The Work of the Pedagogical Patterns Project. *Computer Science Education* 13, 4: 315-330.

Wiedenbeck, S, & V Ramalingam 1999. Novice Comprehension of Small Programs Written in the Procedural and Object-Oriented Styles. *Inernational Journal of Human-Computer Studies* 51: 71-87.

Wirfs-Brock, R, B Wilkerson, & L Weiner 1990. *Designing Object-Oriented Software*. Englewood Cliffs, N.J.: Prentice-Hall.

Authors' Contact Details

Leila Goosen (lgoosen@gk.up.ac.za)
Vreda Pieterse (vpieterse@cs.up.ac.za)

Department of Computer Science

University of Pretoria, Pretoria, South Africa