

An Undergraduate Course in Object-Oriented Software Design

Cathy Bishop-Clark
Systems Analysis Department
Miami University
Middletown, Ohio 45044
bishopcu@muohio.edu

James D. Kiper
Systems Analysis Department
Miami University
Oxford, Ohio 45056
kiperjd@muohio.edu

Abstract: *Many software development organizations are adopting object-oriented methodologies as their primary paradigm for software development. The object-oriented method appears to increase programmer productivity, reduce the overall cost of the software, and perhaps most importantly creates software that promotes reuse and subsequently is easier to modify. Consistent with the change in industry, many universities and industry training organizations are currently in the process of integrating object orientation into their curriculum. There are several approaches including horizontal integration (integrating a small dose of the object orientation into many courses) and vertical integration (having a large dose of the concepts in a single course).*

In 1996, the Systems Analysis department of Miami University opted for the latter approach and added a new course to its curriculum. It is a course that is intended to provide some in-depth exposure to object-oriented design and implementation. It should be of particular value to faculty in computer science and information systems departments (both at the 4-year and 2-year institutions) as well as those in industry training organizations who are looking for ways to incorporate the object orientation into their curriculum. In this paper, we will describe the choices our department made, what worked well, and what needs to be improved.

Previous Research

Several papers have been published which discuss object-oriented courses as well as issues involved in teaching object orientation at the undergraduate level. [4,6,8,9,11]

Perhaps the most important point that is well documented and understood is that paradigm shifts are very difficult and the shift from a procedural paradigm to an object-oriented paradigm is no exception. [6] Such a shift in computer science education requires considerable time and

effort.

The first factor to consider when introducing the object orientation into the curriculum is how and where to introduce the material. Some authors [8] suggest an incremental strategy in which some components of the object orientation are introduced in a variety of courses. For example in CS1 and CS2 courses classes and objects could be introduced. Later, students would learn about object-oriented databases in a database course, object-oriented design in graphics and simulation courses, etc.

Other authors have introduced the paradigm in a single course concentrated on the object orientation. For example, Lattanzi and Henry [4] discuss their experiences teaching the object-oriented paradigm in a senior-level software engineering class. Although they introduced the object-oriented paradigm in a senior level course, they conclude by recommending that the object-oriented paradigm be integrated into the curricula early.

Instructors must also decide whether to use a programming language and if so which one. Some authors [11] suggest that object-oriented design should be taught without using any programming language. They suggest that using a programming language in a course that emphasis design detracts from the design objectives. Their work refers readers to an extensive list of other sources that show the limited cognitive consequences of learning computer programming languages. Another author [8] suggests using a language but making a concerted effort to focus on design and not language issues. She points out that mastering the object-oriented language is not the difficult part of the object-oriented paradigm.

Still, it seems that many of the object-oriented design issues remain too abstract for first and second year students without specific examples in a programming language. Assuming that an instructor chooses to use a language to help illustrate the design concepts, several are available including Smalltalk, Java, Ada, Eiffel, and C++. Each

language has its advantages and disadvantages (see [2] for a detailed comparison of Ada, Eiffel, and C++, and [3] for a discussion of language choice in CS1.)

Curriculum

The Systems Analysis program at Miami University in Ohio consists of 128 credit hours, 45 of which are in the Systems Analysis program. The curriculum integrates course work in computer science, information systems, mathematics, probability and statistics, mathematical modeling, oral and written communication, the liberal arts, and science. The Systems Analysis program begins with typical CS1 and CS2 courses that use the programming language C++. The first course provides an introduction to the fundamental programming concepts and the second provides an introduction to data structures and data abstractions. In the first course, the procedural paradigm is emphasized. In the second course, students use C++ and classes to implement data structures, but the emphasis in this course is on abstract data structures, not the object-oriented paradigm.

In the mid 1990's our department began to see a need to provide students with a substantial experience involving the object-oriented paradigm and we introduced a third programming course which would follow this CS1 and CS2 sequence. This course would include a student team-based experience developing a larger scale project in an object-oriented environment. The majority of the students take the course at the beginning of their sophomore year and were therefore introduced to the object orientation early in their career.

The Course - Object-Oriented Software Design

The new course entitled "Object-Oriented Software Design" provides students with an introduction to object-oriented design and hands on exposure to implementing object-oriented designs. There are two primary threads to the course. The first is object-oriented design. The students study one technique of object-oriented design (specifically they learned the Booch method) and read the book Designing Object-Oriented Software. [12] (For the fall semester of 1997, we have adopted the UML method as described in the text UML and C++. [5])

At the same time that students in this course are learning the principles of object-oriented design, they learn object-oriented programming through the language C++. For this they use the CD "Mastering Microsoft Visual C++ 4.0". [7] By simultaneously studying both object-oriented design and Visual C++, students had a unique opportunity to apply the techniques they were learning. In the process of learning object-oriented design and Visual C++, they also were exposed to CASE tools, various interface and code generators, and testing strategies. They also had the important experience of developing object-oriented software

in a team environment.

The bulk of class time was spent on either object-oriented design or Windows programming in Visual C++ with the Microsoft Foundation Classes. The study of Visual C++ included an overview of the language of C++ and specific examples of the object-oriented features such as inheritance, polymorphism, and encapsulation. Therefore, the students studied object-oriented features at two levels—one independent of the language (in the object-oriented design portion) and the second specific to Visual C++. Since this was the first exposure for many of the student to developing windows, event-driven software, some class time was spent on understanding the particulars of the Visual C++ environment. The instructors of the course created PowerPoint presentations for each of the major topics of the course and used these to guide class lecture and discussion. The PowerPoint lectures were organized in the following categories: introduction to the object-oriented paradigm, review of C++, Visual C++, object-oriented design, and software testing.

Course Objectives

The primary course objectives were to allow students to gain experience in object-oriented design and provide practice implementing their design in a team environment. Some more detailed objectives from our syllabus are listed below:

- to learn some common patterns of program structures that commonly appear in moderate to large pieces of software,
- to learn about programming environments composed of CASE tools, to understand how software tools can aid the development process, and the role of CASE tools and development methodologies,
- to learn to use some software development tools, e.g. project management, on-line debuggers, interface builders and other code generators, source code analyzers, version control tools, etc., and to understand the benefits and limitations of such tools,
- to understand the importance of a software development method, and to gain some facility in at least one, e.g. object-oriented design and programming,
- to learn and experience software development strategies for debugging and testing code, including various testing methods like equivalence partitioning, boundary value analysis, random testing, and mutation testing.
- to experience a moderate sized software development in a small team following a prescribed method,
- to learn some of the basic development principles current paradigms, e.g. event-driven programming,

- to learn the portions of an appropriate programming language that support object-oriented programming, e.g. class inheritance in C++, object-oriented frameworks, interface building in Visual C++,
- to understand the benefits and limitations of reusability and to make use of reusable software components such as function libraries and class libraries,
- to study features and benefits of development environments and frameworks for environments, e.g. class browsers, etc.

Course Assignments

There are several different types of assignments that were used throughout the course. These include homework, class presentations, programming projects, text and outside readings, quizzes and exams. All of this material is available to interested faculty.

Homework Assignments

Homework assignments were given regularly throughout the class (approximately 10 homework assignments in a 16 week period). The homework assignments were relatively short exercises—most of which related to learning Visual C++. For example, in one homework assignment students had to modify an existing Visual C++ assignment to draw in colors. Students were also asked to create object-oriented designs based on a brief one-paragraph description of a problem. These designs were then used in class as a point of discussion.

Presentations

In at least one of the classes, groups of students were required to give a presentation on a topic that was closely related to object-oriented design but not covered in lecture material. In other assignments, students presented other object-oriented Languages (such as Java) and GUI's, and tools for programming in the large. Most students did a substantial amount of learning outside of the class for the presentation.

Programming Projects

Between 2 and 4 programming larger projects were assigned. An example of an initial programming assignment implemented in Visual C++ was a tic-tac-toe game. The program allowed two users to play tic-tac-toe in a graphic, point-and-click manner. The final program project required that an object-oriented design be turned in prior to an implementation. The students took a requirement specification, designed a solution according to the method of object-oriented design prepared in class, and implemented it in Visual C++. For example, in one final project students

were required to first create a design for a system that would maintain and manipulate information (such as name, advisor, courses required, courses taken, etc.) about students in a graduate program. They then used Visual C++ to implement the design.

Homework and Readings

Students were expected to read and study both texts thoroughly and carefully. Students read primarily from two books—one on design and one on Visual C++. Supplementary readings on an introduction to the OO paradigm, and software testing were also given.

Quizzes and Tests

Quizzes were given regularly to assure students stayed current with the material. A midterm and a final exam were given. The questions on both exams were essay in nature.

Selection of Object-Oriented Design Technique

Two of the key choices facing an educator designing such a course are the design technique to be used and the programming language used to implement the design. We have generally tried to stay in the mainstream in our choice of an object-oriented design (OOD) method. In early versions of the course, we based our discussions of OOD on Booch's method. [1] More recently, we have asked students read Wirfs-Brock's [12] book and practiced using the CRC technique. In the fall of 1997, we have evolved to the use of UML [5] and a version of Rational Rose. [10]

There is a plethora of OOD methods. Our criteria for choosing a method have been ease of learning and future utility of design skills learned. Although there is no standard method, we believe that OMT or UML are most likely to emerge as the method of choice. It is true that UML has the disadvantage of being fairly complex. However, we believe that a reasonable subset of this method (as presented in [5]) can be mastered by students in one semester.

Selection of Object-Oriented Design Language

In our case the language used was Visual C++ primarily because it is the language used in the first two courses (CS1 and CS2). While students had used C++, they had not used it to create windows programs or for event-driven programming, nor has they used the Microsoft Foundation Classes. The primary advantage of using C++ is that it gives our students a marketable skill. In addition, C++ supports object-oriented programming (OOP), and improves on C in its input and output, and in the use of reference data

types. Visual C++ version 4.0 (and the more recent version 5.0) from Microsoft has been an effective tool to support the windows, event-driven programming that we want to do. It is easy to demonstrate inheritance, collections, and associations by studying the code that is generated. Students are exposed to graphical interface builders, code generators, and templates. On the other hand, the learning curve for MFC is substantial and the students leave with only a small portion of the package mastered. It was sometimes difficult for students to debug a Visual C++ application [2] because of the complexity of window's programming.

Evaluations

At the completion of each section of the course, students were asked to complete evaluations. Some of the pertinent ratings were to the questions of whether the course was intellectually challenging, and whether they learn a great deal. As can be seen in table 1, the responses to these increased from lower ones in the spring of 1996 to very positive values in the fall of 1996 and spring of 1997 as the professor gained experience in teaching and the pedagogical methods matured and developed. (On this rating form, the top rating of 4 was described as "strongly agree," rating 3 as "agree," 2 for "neutral," 1 for "disagree," and 0 for "strongly disagree.")

Table 1: Course Evaluation Data

Question	semester	rating
Course intellectually challenging	Spring 1996	2.9
	Fall 1996	3.4
	Spring 1997	3.2
Learned a great deal	Spring 1996	2.6
	Fall 1996	3.3
	Spring 1997	3.2

Written comments on these evaluations frequently described that learning Visual C++ for windows programming was one of the strengths of the course. In other comments, student mentioned that they valued learning OOD. For example, one student wrote "The strengths of this course are in teaching students how to design classes for an object-oriented program and how to program in Visual C++."

From the instructors' point of view the course was successful but can be improved. The placement of the course seems quite appropriate. Students are introduced relatively early in their computing career to the object orientation. By their sophomore year, most of students had heard of the object-oriented approach and were interested in learning more about it. Since they had had two semesters' experience in programming in C++, it was logical to continue their experience in C++. This provides students with some depth to a computing language.

The fact that students had been programming for two semesters in a procedural paradigm did not seem to interfere

or conflict with their capability of understanding and using the object-oriented paradigm. In fact, the added maturity seemed to be a benefit in taking a more serious look at design of larger software systems. It is not clear that students with no programming experience would be able to appreciate the importance and value of design.

One of the problems with the existing version of the course is that students do not do a good job of integrating the two threads of the course (object-oriented design and Visual C++). It is difficult to create a manageable project that does involve both a substantial design and is manageable to code in a semester. One instructor gave 2 programming projects, and a final project, which involved only a design and test plan. The students reported that the final project took much more time than the previous programming project. A project that includes both design and implementation would be ideal.

A second issue, which will be resolved in time, is the order of the material and how to present the material. Currently, the majority of the material is presented in lecture form with much class participation. The object-oriented design portion of the class lends itself to much more group work than is currently used. In small groups students can create some small designs and use class time to compare various designs.

Summary

We have successfully taught this course for the last three semesters on two campuses to over 100 students. Coming into the course, many of our students had lost interest in programming because of the intense nature of CS1 and CS2 and the “toy” applications that they had created in those classes. In this course, students have generally regained some enthusiasm about learning to create interesting applications that have pull down menus, dialog box, and graphics. They have been appreciative about the chance to learn how to design larger pieces of software, understanding the increased importance of planning and design as the project become larger. The experience of working in teams generally has been a positive one. This course has had a positive effect on the students when they are enrolled in the course, and afterward as they apply what they have learned in later classes, in internships, or in their professions.

References

2. Booch, Grady (1991). Object-Oriented Design with Applications, Benjamin Cummings.
3. de Vivo, Gabriela, de Vivo, Marco, and Isern Germinal. E Pluribus Unum: OOP Selection. *SIGSCE Bulletin*. 29:2 (June 1997), 17-22.
4. Kiper, James D. and Ken Abernethy. Language Choice for CS1 and CS2: Experiences from Two Universities, *Computer Science Education* 7:1(1996), 35-52.
5. Lattanzi, Mark R., Henry, Sally M. Teaching the Object-Oriented Paradigm and Software Reuse: Notes from an Empirical Study. *Computer Science Education* 7 (1996), 99-108.
6. Lee, Richard C. and William M. Tepfenhart (1997). UML and C++ A Practical Guide to Object-Oriented Development. Prentice Hall.
7. Lilly, S. The structure of software revolutions. *Object Magazine*, 1993, 77-79.
8. Mastering Microsoft Visual C++ 4. Microsoft Press, 1996.
9. Northrop, Linda M. Finding and Educational Perspective for Object-Oriented Development. *Computer Science Education* 4 (1994) , 5-12.
10. Osborne, Martin. Computing Curricula 1991 and the Case for Object-Oriented Methodology. *Computer Science Education* 4 (1993), 25-33.
11. Rational Rose (1997). <http://www.rational.com/products/rose/datasheet.html>.
12. Sims-Knight, Judith E., Upchurch, Richard L. Teaching Object-Oriented Design Without Programming: A Progress Report. *Computer Science Education* 4 (1993), 135-156.
13. Wirfs-Brock, Rebecca, Wilkerson, Brian, Wiener, Lauren (1990). Designing Object-Oriented Software. Prentice Hall.