Object-Oriented Programming for Freshmen Computer Engineers (and Their Professors)

Mark J. Sebern
Milwaukee School of Engineering

Abstract

For practicing computer engineers, the object model has become increasingly important. Recognizing this fact, elective courses in object-oriented programming (OOP) have been offered a number of years. At some point, however, it becomes desirable to integrate this technology throughout the computer engineering curriculum. Such an effort raises many questions, such as language selection and topic sequence. Some faculty members, comfortable with traditional software development techniques, have concerns about making the transition.

At the Milwaukee School of Engineering, we decided to go ahead. Based on input from stakeholders, including our industrial constituency, C++ was chosen as the primary language. Faculty and staff were offered a ten-week course in object concepts and C++, and many chose to participate. As a first step, we converted two freshman software courses from C to C++, and from a focus on structured programming to an object-oriented approach. The first course concentrates on programming basics and the use of existing class libraries, while the second introduces software design and class implementation.

Some lab projects involve the development of traditional stand-alone "console mode" programs from scratch, while others incorporate pre-written graphical user interfaces and class libraries. The latter projects are popular with students, since they resemble contemporary software products. An added bonus is the firsthand experience of the benefits of software reuse.

Introduction

The computer engineering curriculum at the Milwaukee School of Engineering (MSOE) incorporates a series of software courses, beginning with introductory programming and progressing through software design, data structures, graphics, numerical methods, operating systems, software engineering, and other topics. While students use a variety of programming languages in these courses, one primary language has traditionally been emphasized.

The choice of primary programming language has been based on teaching objectives, available technology, and recommendations from various constituencies (employers, industrial partners, and alumni). In recent years, we have focused on the C programming language, with significant success. However, in light of industry and technology trends, we became convinced that a switch to an object-oriented language was desirable.

Preparing for OO

The process of introducing object technology at MSOE was probably not unusual, but may be of some interest to others still contemplating such a move. In the spring of 1995, a group of faculty

met to discuss the issue, and settled on C++ as the most likely candidate language for our environment. MSOE had been receiving numerous requests from our industrial partners for in-plant courses in object-oriented programming (OOP) and C++, and our traditional undergraduate students (many of whom work as interns) were pushing us in the same direction.

For several years, MSOE had offered a senior elective course (CS-481) that served as an introduction to OOP and C++, for students with significant prior experience in C and structured programming. During the summer of 1995, the author taught two sections of this course in-plant at a local medical electronics company, to experienced software developers. Subsequently, demand for the course and for other OOP/C++ offerings continued to grow, into the fall of 1995.

At this point, however, only a small number of faculty had significant experience with OOP and C++. This caused an obvious problem in meeting the demand for course offerings, but also made it difficult for the department to make an informed decision about the adoption of a new "primary" programming language for the computer engineering program.

As faculty interest increased, a decision was made to offer an in-house course, an enhanced version of the traditional C++ elective. During the winter quarter of 1995-1996, about twenty faculty and staff participated. Upon completion of the course, the faculty were polled regarding the introduction of object technology in general, and C++ in particular, into the computer engineering curriculum. The sentiment was heavily in favor, with a significant number of faculty expressing a willingness to teach the new and modified courses.

The Freshman Courses

Since the MSOE schedule is based on quarters rather than semesters, the first two software courses do not correspond exactly to the common "CS1/CS2" sequence. Some of the more advanced data structures topics are covered in a sophomore course (CS-285).

One major question in designing courses in object technology and C++ is when to introduce the concepts of objects and classes. Some suggest introducing these topics late, after treating traditional procedural programming, while others argue that these key concepts should be introduced as early as possible.

We had heard from other schools that, although faculty sometimes had to struggle to switch to an object-oriented paradigm, students who started out with objects had little difficulty with the concepts. For this reason, we chose a variation of the "objects early" approach, and adopted a textbook that embodied this philosophy[3]. Since we felt that the Standard Template Library (STL)[4] is a critical part of the evolving ANSI/ISO C++ standard, we incorporated additional material on simple uses of STL containers and iterators. For the same reason, we decided to stress a string class library and to de-emphasize character arrays (C-style strings), especially in the first course (CS-182).

The result of this planning was the set of course topics shown in Table 1. We were pleasantly surprised to find that, with the availability of the STL and C++ reference objects, we were able to delay the introduction of pointers until halfway through the second course. In the prior C-based

courses, we found that students had the most trouble with character arrays and pointers, especially when they were first attaining proficiency in programming.

CS-182 Computer Programming	CS-183 Software Design
Introduction to OOP	Structures & classes
Basic C++ syntax	Class members: data & function
Simple stream I/O	Simple list class: array implementation
Functions & arguments	STL **`list`** and iterators
Using class libraries	Overloaded functions
Selection & iteration	Operator overloading
C++ arrays & STL **`vector`**	OO design: classification & use cases
String class usage	Pointers & dynamic objects
	String class implementation
	Composition & inheritance
	List class: linked list implementation
	Polymorphism: templates & virtual functions

Table 1. Freshman Course Topics

Lab projects

In an initial programming course, it is sometimes difficult to keep students interested, because they do not yet have the skills necessary to develop programs of significant complexity. In some of the CS-182 labs, students have the opportunity to integrate their own code into an existing software structure. For example, as soon as functions are introduced, students are asked to develop a function that implements the equations of motion for a projectile. Each student's function is then added to an existing application shell; the result is shown in Figure 1.

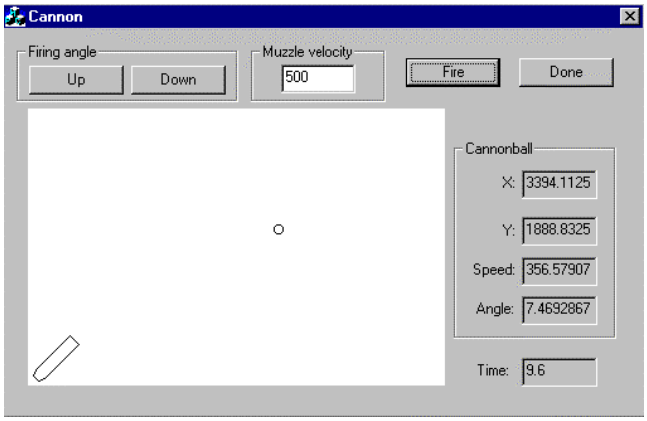

Figure 1. Projectile Trajectory Program

In addition to producing a more pleasing result, the graphical display of the projectile motion makes it easy for the student to see when there is an error in the calculation, such as mixing up radians and degrees in the angle of the velocity vector.

After the introduction of class libraries and use of member functions, students develop an animation using a number of predefined classes, as shown in Figure 2. The application framework takes care of timing and the actual graphic rendering of the objects created by the students.

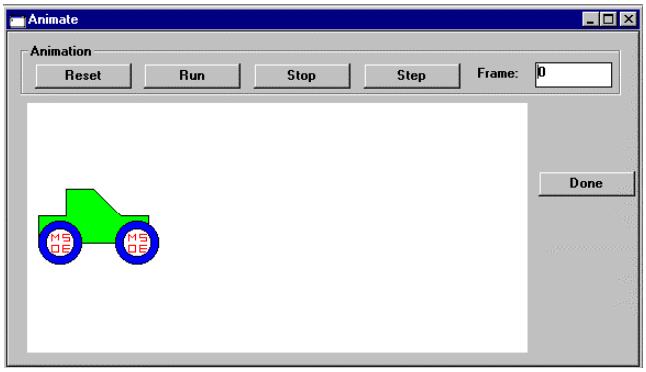

Figure 2. Animation Program

To develop skills in implementing decision behavior (selection statements and operators), students write a set of functions that control an object as it navigates a randomly generated maze, as shown in Figure 3. As students build their programs, they also discover the importance of maintaining system state; if they fail to do so, the result is a clearly visible repetitive motion as the object gets "stuck" at the top or bottom of one of the maze's columns.

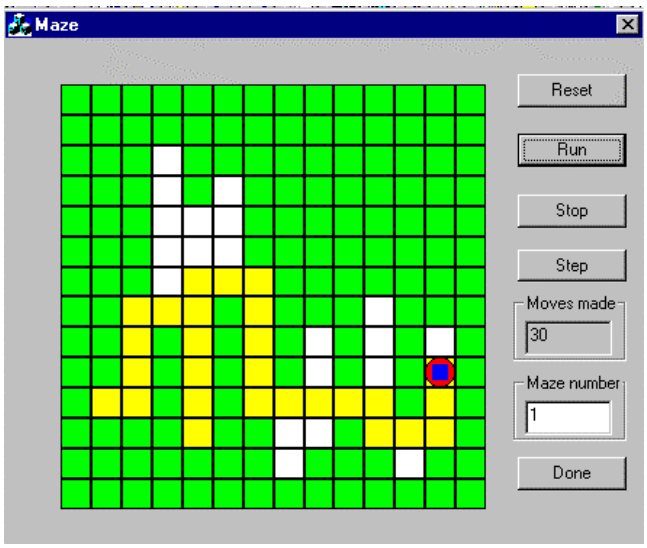

Figure 3. Maze Program

In working on the lab projects that required them to work with pre-existing software, students found out what it is like to write to a specified interface. Some initially viewed this as a limit to their creativity, but later realized how important this skill is when working as a team member.

A number of students also commented on how much more they could accomplish with a reasonable amount of effort, if they were willing to build on software that others had written

(such as the application frameworks provided to them). It is hoped that this experience will make them more likely to take software reuse seriously, both as students and as practicing professionals.

Object-Oriented Design

Although a primary focus of the freshman courses is object-oriented programming, we wanted to make sure than object-oriented design (OOD) was also introduced, particularly in CS-183. In later courses, these students will use computer-aided software engineering (CASE) tools (e.g., Rational Rose) that embody design techniques like those described by Grady Booch[1]. In CS-183, we stress two basic concepts: classification and use cases.

Classification is the process of identifying classes and objects, given a preliminary problem specification. Students are taught to look for the nouns, qualifiers, and verbs in the specification, with an eye to choosing classes and defining their attributes and behaviors. They also learn the roles of two different kinds of objects, those corresponding to things in the application domain and those that are artifacts of the software system itself.

Use cases, as proposed by Ivar Jacobson[2], are sets of scenarios of proposed system operation. By considering concrete instances of system interactions, the students are able to develop and enhance the set of classes that will be the basis for system implementation.

One advantage of the more complex projects that are made possible by C++ and the STL is that students are more likely to experience firsthand the benefits of doing a careful design before starting the implementation. In the process, some find out the hard way that a complex design will probably not result in a simple implementation, and that modifying a design in the early stages is a good deal easier than changing all the code later.

Summary and Future Directions

Student reaction to the new freshman courses has been very positive. Because some low-level details can be deferred and higher-level constructs are available, most students are able to complete more complex programming tasks than could their prior counterparts using procedural design and the C language.

Along the way, we found that we had to develop a fair amount of supplementary material, much of it posted on MSOE's Web server for easy access and updating. The textbook, for example, discusses templates but not the Standard Template Library, and does not introduce a string class (to replace character arrays) as early as we would like.

Dealing with a variety of C++ compilers was also a challenge, but we didn't want to discourage the students from working on their own systems. For this reason, all labs were developed to run on at least two different commercial compilers. The lab files were distributed from the Web server as well.

In using the lab materials, we found a number of rough edges that need to be taken care of. For most labs, a single professor was responsible for developing the software and instructions. Very

often, one of the other professors would review the lab and suggest changes or prepare a document with clarifications and further explanations of points that were unclear. All this material was then made available to the students. We plan to incorporate the revisions into the original materials before the next offering of each course.

The freshman courses are, of course, only the beginning. We plan to modify the data structures course to address C++ and the STL, and hope to exploit polymorphism and virtual functions in the junior graphics course (CS-321).

We are already starting to see more references to the object model and increased use of C++ in other courses. In part, this is because more faculty now feel comfortable with these topics. As other students find out what is happening in the new courses, many of them are starting to learn C++ and object concepts on their own.

Some course materials discussed in this paper are available on the World Wide Web, at `http://www.msoe.edu/~sebern/asee97/oop/`.

Acknowledgments

Lab facilities for CS-182 and CS-183 are supported in part by a software license grant from Microsoft Corporation's Instructional Lab Grant program. Inspiration for the animation program (Figure 2) came from an earlier DOS program developed by Professor Sue Fitzgerald of Metropolitan State University. The concept of the maze program (Figure 3) was suggested by prior work of Professor Richard Rasala and his colleagues at Northeastern University.

References

1. Booch, G. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, Redwood City, CA, 1994
2. Jacobson, I., Christerson, M., Jonsson P., and Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.
3. Perry, J. E., and Levin, H. D. *An Introduction to Object-Oriented Design in C++*, Addison-Wesley, 1996.
4. Stepanov, A. and Lee, M. *The Standard Template Library*. Hewlett-Packard Co., Palo Alto, CA, 1995.

Biography

MARK SEBERN is a professor in the EECS department of the Milwaukee School of Engineering, and director of the Bachelor of Science in Computer Engineering program. He received Ph.D. and B.S. degrees in electrical engineering from Marquette University in 1974 and 1972, respectively. Prior to joining the full-time MSOE faculty in 1994, he worked in industry for twenty years as a computer engineer and consultant.