AN OBJECT-ORIENTED INFORMATION SYSTEMS COURSE

Dr. T.M. Rajkumar
Department of Decision Sciences
School of Business Administration
Miami University
Oxford, OH 45056

ABSTRACT: An Object-oriented information systems course taught at the senior level is described. C++ is used as the language to teach object-orientation. In addition to the language, analysis and design concepts are integrated into the course. A topic outline for the course is described. Course materials used to teach such a course and the projects assigned are presented. Concepts that need to be stressed such as encapsulation, inheritance and polymorphism are reviewed. The impact of OO approaches on databases, software development are discussed. The teaching of C++ in a DOS based environment as opposed to using GUI based methods is briefly discussed.

KEYWORDS: Object-Oriented Programming, Object-Oriented Analysis, Object -Oriented Design, C++, CIS Curriculum

INTRODUCTION

In the last few years, there has been growing enthusiasm, for object-oriented (O-O) approaches to information systems. There have been significant advancements in all areas of object-oriented information systems from programming to analysis, design and development.

Acceptance also has come in the business world where commercial applications in banking, CASE, and CAD/CAM are being developed using object-oriented techniques. For example, Brooklyn Union gas has developed standard business applications such as billing using O-O approaches [1]. Since there are currently some 5000 programmers in various companies developing systems based on object technology, it can be concluded that this technology is both proven and mature [2]. It is therefore essential that this technology be introduced and taught in the MIS curriculum.

There are at least two ways in which object-orientation can be taught within the MIS area. The first is to teach a little bit of object-orientation in every MIS course; for example Cain [3] proposes that the systems analysis and design course introduce object-oriented analysis (OOA) and design. The COBOL course can introduce object-oriented programming (OOP) terminology while teaching related programming topics such as variable types and subprograms.

The database course can then introduce object-orientation with respect to databases and semantic data modeling. Though this is a valid approach, the student sees object-orientation in bits and pieces and does not see the overall benefits and structure of object-orientation. In addition, concepts such as encapsulation, inheritance etc. have to be reintroduced in each course. This leads to unnecessary repetition.

A possibly better approach is to teach object-orientation concepts in a separate course. Such an approach would allow the concepts related to object-oriented information systems to be presented once, and allow comparisons with traditional structured approaches to be made. A more compelling reason is that the student is better able to appreciate the benefits of object-orientation once the student has analyzed, designed and developed a system using the object-oriented approach. The aim of this paper is to present the structure of such a course and elaborate on the concepts that need to be stressed. The next section discusses the outline of such a course.

COURSE OUTLINE

Students enrolling in this course should ideally be seniors and have had at least the COBOL, Database, Systems Analysis and Systems Design courses. Such prerequisite allows one to structure the course into the following major sections:

Introduction: (1 week)

In this section the common concepts of object-orientation such as what is an object, methods, and features such as encapsulation, inheritance, and polymorphism can be introduced to the student.

Programming: (8 weeks)

In this section of the course the student is introduced to an object-oriented language. This is necessary so that the student can implement some sample object-oriented systems. The author has used C++. For better or worse, C++ is the object-oriented language of choice among software developers in the industry [2]. The first 4 weeks of this section is used to bring the students up to speed in C, and the second 4 weeks can be used to introduce the object-oriented aspects of C++ in specific. The course does not assume knowledge of C and it is essential that the students know C to be able to program in C++, since the object-oriented features of C++ are built as extensions of the C language.

Analysis and Design: **(4 weeks)**

In this section of the course, the student is introduced to various analysis and design techniques that use object-orientation. The limitation of approaches that use structured methodology for analysis and design is first discussed. Newer approaches such as Yourdon and Coad's analysis method and Booch's design method are then taught. Since, there is no agreement currently in the literature that the above two methods are the standard methods used, other techniques such as the responsibilities and collaboration approach of Wirf-Brock et al [4] are identified and very briefly discussed. The effect of object-orientation on the maintenance and other phases of the life cycle is also brought out.

Management Issues: (1 week)

In this section, issues such as education, training, project management, and difficulties are discussed.

A drawback to the above approach is that the student is not exposed to the object-oriented database perspective. This is difficult to achieve within the 14 week span of a semester. If C did not have to be taught, and the instructor has to only teach C++, then it may be feasible to bring in the object-oriented database issues. The above outline is summarized in Table 1 and has been used by the author successfully to teach the OOIS course.

COURSE MATERIALS

Language

There are a wide variety of O-O languages such as Simula, Smalltalk, Eiffel and C++ in which O-O development can take place. Even language independent/ neutral approaches to object development such as SOM from IBM are available [5]. The last few years have been marked by the increasing popularity and acceptance of C++ by a large group of programmers.

C++ is used in thousands of projects in industry [2,6]. Because of the predominance of C++ in industry it is essential to teach OOIS through the C++ language. The author has used Turbo C++ for DOS as the specific language of choice. Other choices exist such as Borland C++, or Microsoft C++ or even the freely available GNU G++. Turbo C++ has been used because it makes minimal demands on the hardware required to run the system (386SX, 1MB RAM and occupies 6 MB of hard disk space). The others make a more hefty demand on the computer and can be run efficiently only from the network (For example, Borland C++ requires 20 MB of disk space and Microsoft C++ at least 14 MB).

Table 1: COURSE OUTLINE

Topic	Concepts	Time
Introduction	Basic Concepts	1 week
Programming	C	4 weeks
Programming	C++	4 weeks
Analysis	Yourdon and Coad's Methodology	2 weeks
Design	Booch's Methodology	2 weeks
Management	Managing O-O Technology	1 Week

Books

There are very few good C++ books on the market that are oriented towards the business student. In addition, most books assume a working knowledge of C and are oriented to help move the C programmer to program in C++. The author uses a book by Chirlian [7].

This book teaches C++ from the basics, and covers C++ with a business oriented example (banking). The initial parts of the book is not very business oriented and students have some difficulty with the initial chapters (some of the C concepts). Other good books that teach C++ from the beginning and can be used are those by Ammeraal [8] and Lippman [9].

For the analysis and design aspects, two books are recommended. These are books by Yourdon and Coad [10] and Booch [11]. These are recommended because they match the techniques that are taught in the course. Additional books such as those by Shlaer and Mellor [11] and Wirfs-Brock [4], are also appropriate. As developments in this area are rapidly taking place, this section of the course needs to be supplemented with readings from journal articles.

IEEE Computer, IEEE Software, Communications of the ACM, among other journals; carry summary and research articles in this rapidly progressing field. Rather than have the student read the article, the author has provided summary papers for the students to read.

In the management section entire articles are assigned for reading from current periodicals such as Datamation. In addition, the instructor can bring in his/her perspective and have the student present

their experiences with projects that they are doing for the class.

Projects

Projects are assigned throughout the course. As the course is fast paced and the work load great, all projects are assigned as team projects. The programming projects are given mainly from the exercises in the books. In the C part of the course student teams are required to convert a COBOL program to C. This is to emphasize the differences in approaches and allow comparison between the two languages. In addition, before the C++ part of the course begins, students are required to turn in a design (not code) for a banking / ATM application.

This enables each student to appreciate the design differences between regular design and object-oriented design. The banking example is chosen because the Chirlian book uses the banking example to demonstrate C++. The student teams are made to recode a C project using C++ and object-oriented techniques to further crystallize the differences in approaches. Furthermore a final project (an application program of their choice) is required of all student teams.

For the analysis and design parts of the course, little is required to be turned in except a few exercises from the book. The students are however required to turn in their analysis and design with the final application project assignment. Students have the hardest time in analysis and design part of the course. The thought process used here is different from that used in the traditional structured approaches. In retrospect, increased attention has to be paid to this phase of the course.

In teaching object-oriented programming, analysis and design, emphasis in the course is not placed on the syntax of the language or diagramming tools; but on the concepts of object-orientation. Emphasis is placed on why these concepts are important and how they differ from traditional languages and techniques. The pitfalls that must be avoided and the difficulties with the approach are stressed. The next section briefly discusses some of the concepts that are emphasized.

In teaching object oriented programming, analysis and design, emphasis in the course is not placed on the syntax of the language or diagramming tools; but on the concepts of object orientation.

CONCEPTS

Even though object-orientation has been around for many years there is still no unique definition of the term object. Object-orientation has been applied in different areas of information systems such as programming, artificial intelligence, operating systems, and database systems. Similar concepts have been called different terms in each of these areas and the same term may be used with different meanings [13]. For some standardized definitions and discussions of the various approaches refer to Snyder [13]. In general, it is agreed that three of the concepts: encapsulation, inheritance, and polymorphism are important. These are presented next.

Encapsulation is the ability to gather into one place all aspects related to a given abstraction of a real world object [14]. In C++ this means embodying both data and functions within an object through the mechanism of a class. The advantage of encapsulation is that we can modify the implementation while the calling programs that use the class remain unaware of the change. This is possible because implementation is separated from the interface. Encapsulation makes possible information hiding by representing the internal representation of a class type through private variables and functions. If we consider a pixel on a screen as a class, its location and color properties can be represented with variables such as x, y, and color. Functions such as get-x, set-x, get-color and set-color can be used by the pixel class to retrieve and set its properties. Color may be internally implemented using the red, green, blue technique or using the hue, saturation and value technique. The user of the class is oblivious to the implementation. Any change made internally to the representation of color is immaterial to the external classes. The color of the pixel is then said to be encapsulated within the pixel class. A diagram representing the pixel class and its interface is shown in Figure 1.

Objects communicate by invoking the functions in the classes. Hence, O-O analysis provide tools for depicting communication between objects. The most important difference between O-O analysis and structured analysis methods stem from the encapsulation notion [15]. Structured analysis methods provide tools to create functional decomposition of operations and to model end to end processing sequences. A functional decomposition violates encapsulation since operations can directly access the data of many entities and is not subordinate to any one entity. This view is not consistent with the O-O notion of encapsulation. Hence, O-O analysis technique provide new tools to model operations as belonging exclusively within the object [15]. In a requirements specification using classes, nouns represent candidate classes, and verbs represent the methods or object behavior. By comparing the methods description for broad classes, other abstract classes can be identified.

O-O design may be defined as a technique which, unlike functional design bases the decomposition into modules on the classes of the objects and not on what function the system performs [16]. In contrast, data flow analysis requires designers to first ask the question what the system does. O-O design tries to avoid such questions and poses a different perspective on the designer.

Inheritance enables objects to share resources by specifying subtype relationships. Classes can be organized in a hierarchy and a derived class can inherit selected data members and member

Figure 1: ENCAPSULATION

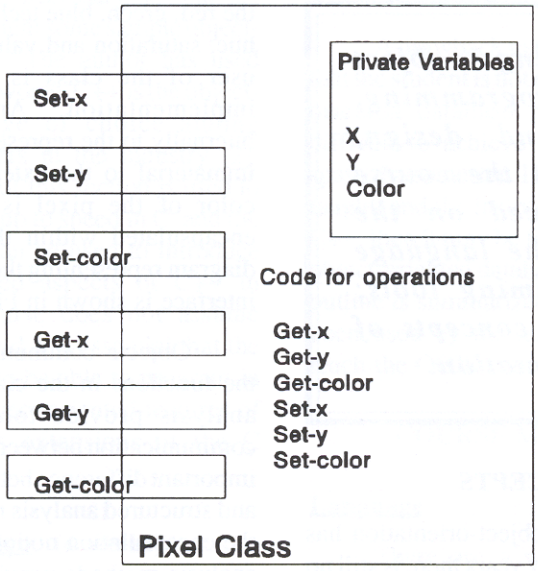
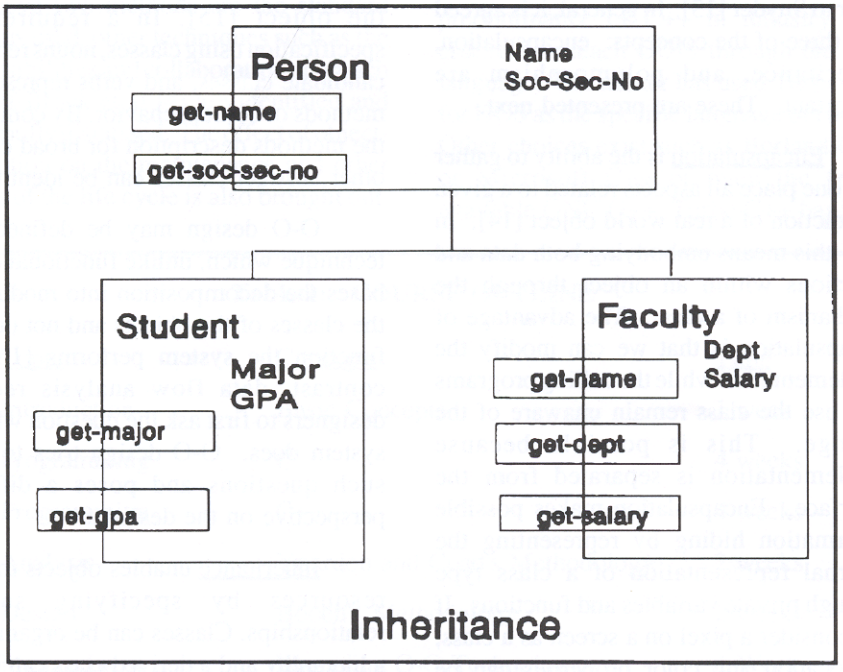


Figure 2: INHERITANCE



functions from higher classes. Derived classes can add new data declarations and functions. For example, consider a person class with name and social security number as data. A student class can be declared to inherit these properties and in addition declare the major and gpa data values. A derived class can also redefine other functions allowing customization to take place. For example, the faculty class can inherit the properties from the class person but in addition can redefine the get-name method to allow customization for faculty. Figure 2 shows the inheritance diagram for the person, student, faculty classes.

Sharing the implementation through a hierarchy results in improved maintenance by reducing source code duplication and reduced executable code size [13]. In the analysis phase, object diagrams provide an object classification and identifies potential inheritance relationships [9]. In the design process, it is necessary to identify where in the class hierarchy the specific class should be located to maximize reuse. The location of the class in the hierarchy depends on answers to such questions as: 1) Is this part of the user interface; 2) Does the class have similar behavior to other classes in the system? [17]. The heuristics of placing classes are weak, iterative and often nonoperational [18]. An additional difficulty with inheritance is that maintainers of O-O code often have to trace through chains of dependencies created by the inheritance [19].

A derived class in C++ that inherits from two or more base classes exhibits multiple inheritance. For example, a graduate student who is a teaching assistant can derive data and members from both a faculty class and a student class. It leads to potential problems. For example; if there is a return_address function in both the faculty class, and the student class; the graduate student class may not be able to identify the return_address function that it should use. In addition; new features are usually a complex synthesis rather than a simple combination of inherited features [20]. Hence; it is not easy to create uniform mechanisms that usefully combine the methods of multiple inheritance classes

[21]. Multiple inheritance is riddled with potential problems and should be used judiciously [22].

Polymorphism

Polymorphism enables a function name to be shared within a class hierarchy allowing each class in the hierarchy to implement the action in a manner appropriate to itself.

Polymorphism is implemented in C++ through virtual functions. Virtual functions enable specification of many versions of the same function through a class hierarchy. The particular version of the function to be run is determined at run time. For example, a bank may provide both commercial and personal loans. The payments that are due on the two types of loans may be calculated using different algorithms due to differing requirements. In such a case, the get-payment function is designed as a virtual function as shown in Figure 3. The desired algorithm for this function is determined at run time based on the type of loan object.

This feature of selecting the code to perform the function at the time the function is invoked is called dynamic binding. This concept is important during the design phase since virtual methods allow for class customization. When building a class, the class designer should see which characteristic might be changed in a derived class [22]. These functions can then be embedded in virtual methods. Polymorphism can also lead to difficulties in maintenance if the designers do not always use polymorphism consistently so that all methods for a particular function do similar things [19].

In addition to these three concepts, a number of newer concepts are being implemented in C++. These include concepts such as templates (generic code) and exception handling. Templates are similar to macro expansion facilities. They allow the user to define the "shape" of a function or class and leave the specifics of the implementation to the compiler. Templates can be declared on functions and classes. Function templates can be used to describe algorithms defined on a wide variety of types. By specifying the type when the program is being written, the algorithm for that type can be generated by the compiler. Class templates are useful to define generic classes such as a linked list. By defining the parameter to the template to be an application class such as an employee, the programmer can generate a linked list of employees. Templates are useful for reuse, as similar code can be generated for multiple objects. In the design phase this leads to choosing and using the right template as this instantiates the code the object would use.

Commercial libraries using the template approach are yet to appear. Exception handling provides a facility for dealing with difficult situations such as failed initialization (constructors and destructors in C++) and error handling in overloaded operations. However, there are no compilers currently that support this in C++. Analysis and design issues surrounding templates and exception handling are still in the research phase.

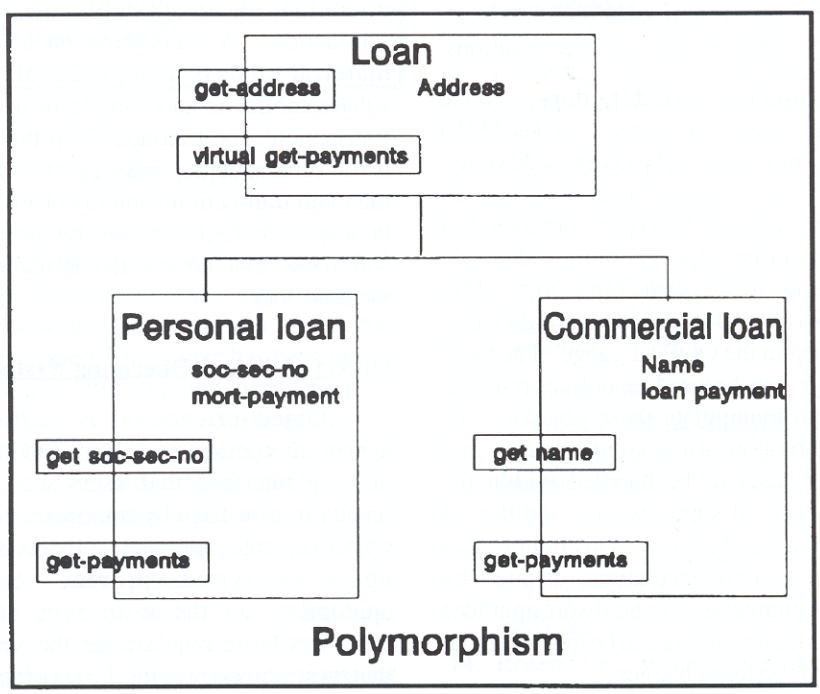
IMPACT OF OBJECT-ORIENTATION ON IS

Object-oriented approaches are present not only in programming, but in all the traditional areas of information systems. Among others, O-O approaches affect 1) Software development process, 2) Database management Systems and 3) Operating systems. These are discussed briefly in this section.

Software Development Process

Object-orientation implies serious rethinking of the software development process [23]. In the classical approach, the phases of design, implementation test are separate steps to be executed in sequence on the whole system. In OOP, the approach is different. The design-implementation-test applies to the life cycle of the individual classes but not necessarily to the life cycle of the project as a whole. This departure is a consequence of the reusability principle which necessitates the designer to rely on

Figure 3: POLYMORPHISM



classes that have already been implemented and tested before higher classes are tested. Hence, a bottom up approach is preferred for object-oriented development with general purpose utility models being built first. Specific modules are built last.

Generalization also implies that the object classes have to be built so that the classes can be inherited and reused in other applications. Such generalization costs additional money, and funding has to be appropriated not only from the current project but the next few projects. This necessitates a shift in emphasis for management from a short term to long term emphasis, profit to investment, program to system, program elements to software component. These shifts are summarized in Table 2.

Table 2: SHIFT REQUIRED IN MANAGERIALEMPHASIS IN THE SOFTWARE DEVELOPMENT APPROACH

Traditional Approaches	Object-Oriented Approach
Results	Tools/Libraries
Department	Organization
Profit	Investment
Project	System
Short Term	Long Term
Program Elements	Software Components
TopDown	Bottom Up

Database Management Systems

With multimedia objects such as sound, image and video becoming more prevalent, database management systems based on O-O approaches that can store, query and retrieve these types of objects are being developed. In addition, a combination of these data types can be present in a complex object. A complex object such as a slide in a class room presentation system or a geographic information system may contain text, graphical images, and sound annotations.

Object-oriented data base management systems (OODBMS) manipulate complex objects, provide views, concurrency control and recovery on objects. OODBMS support some kind of an object-oriented programming language. The data definition language, data manipulation language etc. are then imbedded in the OOP language. The OOP language can be used to define, retrieve, store and manipulate these objects. The OOP language allows inheritance and polymorphism to be handled within the data model. It supports easy addition of new objects and classes. This implies that changes to implementations dictated by the environment such as hardware upgrades or performance tuning can be made without requiring system changes.

Views in OODBMS provide generalizations on the concept of abstraction and encapsulation. Views hide the system implementation, provide the interface and present the information in terms of objects and relationship that occur in the users applications.

Concurrency and recovery mechanisms in OODBMS operate at the granularity of the objects in the users model rather than at the level of files or records. This is important as OODBMS support cooperative transactions which include long and nested transactions. A transaction that allows multiple reviewers of a journal paper to add their comments to the same document is an example of a long transaction. A transaction on a bill of material that is composed of other subassemblies is an example of a nested transaction. The transaction on the lower level subassemblies may commit before the main transaction commits. OODBMS handle such requirements by providing soft locks that inform the initiator of a violated lock.

Object-Oriented Operating Systems

Object-orientation is starting to appear in operating systems and on the desktop interface that users see. This results in user friendly interfaces. In the object-oriented approach, objects in the operating system provide services appropriate for the abstraction. Client processes issue requests for the services that these objects provide. Users of such O-O operating systems do not have to remember the application to run to manipulate the object. They can just click on the object, and the operating system will determine the appropriate program to execute.

Objects are classified in O-O operating systems in terms of the services they provide. The services provided by the objects are available at the interface as menu choices. Users can then visually determine the set of applicable operations on an object by selecting the objects menu. Only the applicable operations are available. The rest of the choices are either dimmed or not present.

Traditionally operating systems provide file handling support, and distinguish between executable and data files. In the object-oriented approach this distinction vanishes. The application and the data of the application merge and become one. This allows applications to be embedded within one another. For example, one can embed a spreadsheet within a word processing document and vice versa; even if the two products come from different vendors.

COMPARISON OF C++ TO OTHER GUI BASED O-O TOOLS

This course has emphasized text based programming in a DOS environment. This approach makes minimal demands on computing resources and teaches OOP in an environment (DOS) that students are most used to. This approach helps keep the student focused on OOP and eliminates the

learning of additional syntaxes and commands of a new application programming interface.

Graphical user interfaces (GUI) such as Windows, Macintosh, and Presentation manager are becoming the standard today. Developing even a simple program such as a "Hello World" program in a Windows environment may take up to 100 lines. The greater part of the code has little to do with displaying the "Hello World" text but is code to manage the screen and write to the Windows application programming interface (API). In addition, programs in these environments are event driven, which means that the code will be divided into many small sections to handle specific events such as a mouse click, rather than the linear sequential programs that the student is used to. In the absence of any tools, learning the windows API and event based programming techniques would be an unnecessary burden in teaching this course. However, GUI based environments such as Actor, Visual Basic and Visual C++ help develop Windows applications and event based programming without the need to learn the Windows API.

Visual Basic provides a forms/tools oriented environment to develop programs. It allows us to draw like a paint program the windows, buttons, list boxes etc. on the screen. These objects can be customized for the specific application. Code can be attached to these objects to respond to events such as mouse clicks and key presses. Visual Basic includes support for object linking and embedding (OLE). This allows applications generated in Visual Basic to use data from any other application in Windows that support OLE.

In addition, the database features of the language allow easy development of client-server applications. The ease of use of Visual Basic enables a user to generate a Windows programs that may take up to an hour in C++. Unfortunately, Visual Basic is not truly object-oriented. Objects in Visual Basic have the notion of properties and methods. The methods are used to access the properties of objects. However, objects in Visual Basic do not support either inheritance or polymorphism. Hence, while it is very useful to develop graphical interfaces and client server programs, it is not really useful to teach object-orientation concepts.

Visual C++ addresses this weakness in Visual Basic by providing a graphical environment to an object-oriented language. Visual C++ provides App studio a collection of resource editors for dialog boxes, menus, cursors etc. This environment generates C++ code for the dialog boxes, menus etc. that can be edited to suit the application. In addition, it has a collection of pre-built class library containing classes of objects that can be customized and reused in the application. Usage of this class library can reduce the amount of code needed for a basic Windows program and developing graphical interfaces. Using Visual C++ in an object-oriented information systems course is a viable alternative. This was not used by the author as the software was not available at the time this course was taught.

... GUI based environments such as Actor, Visual Basic and Visual C++ help develop Windows applications and event based programming without the need to learn the Windows API.

CONCLUSION

An outline for an OOIS course as taught by the author has been provided. Rationale for the specific topics and the order in which they have been taught has been provided. The course has been one of the most challenging and stimulating the author has ever taught. Students coming out of the course have generally been very positive about the course (though they complain about the workload). Students also have the satisfaction that they have learned a very practical subject at the cutting edge of technology. Students leaving the course also realized that this is a field that is still evolving and that continual updating of the knowledge a necessity.

REFERENCES

1. Davis, J., Morgan, T. "Object-Oriented Development at Brooklyn Union Gas", IEEE Software, Jan. 93, pp. 67-74.
2. Jacobson, I., "Is Object Technology Software's Industrial Platform?", IEEE Software, Jan. 93, pp. 24-30.
3. Cain, W.P. "Object-Oriented Programming and the CIS Curriculum", Journal of Information Systems Education, Vol. 3, No. 1, pp. 2-7.
4. R. Wirfs-Brock, B. Wilkerson, and L. Weiner, "Designing Object-Oriented Software", Englewood Cliffs: Prentice Hall, 1990.
5. OS/2 Technical Library System Object Model Guide and Reference, Document S10G6309, IBM CORP, Armonk, NY, 1991.
6. Stuffman, H., "Towards a Less Objected Oriented View of C++", Dr. Dobb's Journal, Dec. 1992, pp. 35-39.
7. Chirlian, P.M. "Programming in C++", Columbus: Merrill Publishing Company, 1990.
8. Ammeraal, L., "C++ for Programmers", New York: John Wiley, 1991.
9. Lippman, S.B., "C++ Primer", Reading, MA: Addison Wesley, 1991.
10. Yourdon, E. and Coad, P. "Object-Oriented Analysis", Englewood Cliffs: Prentice Hall, 1991.
11. Booch, G. "Object-Oriented Analysis and Design", Benjamin Cummings, 1992.
12. S. Shlaer, and S.J. Mellor, "Object-Oriented Systems Analysis — Modeling the World in Data", Englewood Cliffs: Yourdon Press, 1988.
13. Snyder, A. "The Essence of Objects: Concepts and Terms", IEEE Software, Jan 93, pp.31-42.
14. Rosen, J.P. "What Orientation Should ADA Objects Take",

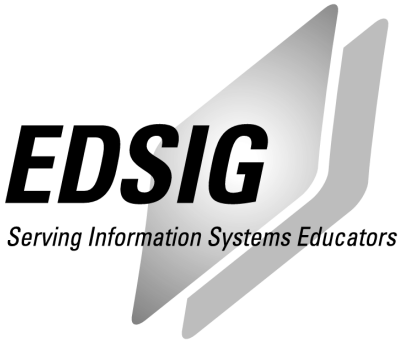
Communications of the ACM, Nov. 92, Vol. 32, No. 11, pp. 71-82.

15. Fichman, R.G., Kemerer, C.F., "Object-Oriented and Conventional Analysis and Design Methodologies." IEEE Computer, Oct. 92, pp. 22-39.
16. Meyer, B., "Reusability: The Case for Object-Oriented Design", IEEE Software, Mar. 87, oo. 50-64.
17. Lorenz, M., "O-O Development Methodology", Englewood Cliffs: Prentice Hall, 1993.
18. Monarchi, D.E., and Puhr, G.I., "A Research Typology for Object-Oriented Analysis and Design," Communications of the ACM, Sep. 92, Vol. 35, No. 9, pp. 35-47.
19. Wilde, N., Mathews, P., and Huitt, R., "Maintaining Object-Oriented Software", IEEE Software, Jan. 93, pp. 75-80.
20. Wegner, P. "Dimensions of Object-Oriented Modeling", IEEE Computer, Oct. 92; pp. 12-20.
21. Taylor, D.A., and Cargill, T. "Object-Oriented Technology", Reading: Addison Wesley, 1991 p. 59.
22. Sessions, R. "Class Construction in C and C++", Englewood Cliffs: Prentice Hall, 1992.
23. Mandrino, D. and Meyer, B. "Advances in Object-Oriented Software Engineering", Englewood Cliffs, Prentice Hall, 1992.

AUTHOR'S BIOGRAPHY

T.M. Rajkumar is an Assistant Professor of Information Systems at Miami University. He obtained his Ph.D. from Texas Tech University. He is the DPMA faculty advisor at Miami University. His research and teaching interests are in multimedia, CASE and object-oriented technology.

Information Systems & Computing
Academic Professionals



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096