BRINGING OBJECT-ORIENTED PROGRAMMING INTO THE UNDERGRADUATE COMPUTER INFORMATION SYSTEMS CURRICULUM

Dr. John K. Gotwals
Dr. Mark W. Smith
Computer Technology Department
Purdue University
Knoy 242
West Lafayette, IN 47907

ABSTRACT: Object-oriented programming (OOP) is becoming the programming methodology of choice in the 1990s. In this paper a justification for including object-oriented programming in the undergraduate CIS curriculum is presented. The paper briefly describes what OOP is, why it is important, and what recommendations two computer related model curricula (DPMA and ACM/IEEE-CS) make about OOP. The course objectives for an introductory OOP course are described. Several potential OOP programming languages are mentioned, and the rationale for the selection of C++ is presented. The paper concludes with a brief discussion of two possible programming environment possibilities: Borland's Turbo C++ and Microsoft's Visual C++.

KEYWORDS: Object-Oriented Programming, OOP, C++, CIS, Curriculum

INTRODUCTION

There is a revolution going on in the software development industry, and part of this revolution concerns object-oriented techniques. Object-oriented became a software related buzzword of the late 80s, and it has now evolved into an accepted technology that has recognized benefits for the software development process.[1] One management consultant reported that Electronic Data Systems (EDS) compared two programming teams, one using object-oriented techniques and the other using conventional methods. EDS found that the team using the object-oriented approach had much better productivity than the team using conventional methods.[2] The computer industry trade press is replete with articles about Object-Oriented X (where X can be Programming, Database, Analysis and Design, etc.). Some of these articles include scare headlines such as "Plan now to prevent career dead ends for COBOL programmers."[3] Indeed, at least one university CIS department is seriously considering the eventual removal of COBOL programming instruction from its curriculum.[4]

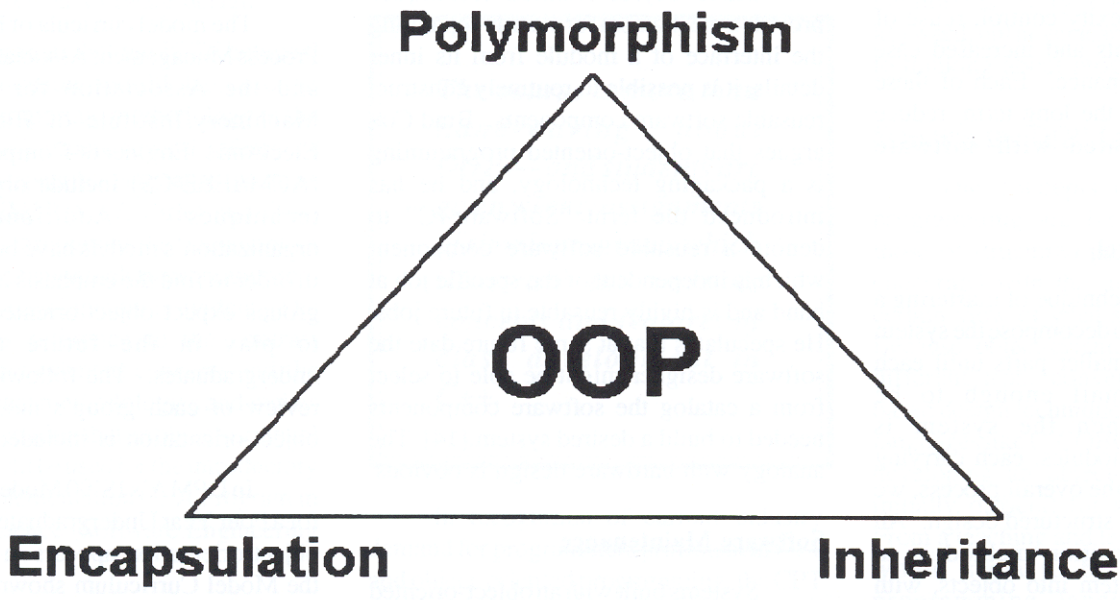
The purpose of this paper is to give an overview of object-oriented programming (OOP) and its current significance, to summarize the extent to which OOP is included in current or proposed model curriculums, and to describe what one CIS department is doing about the situation.

THE CRISIS IN SOFTWARE DEVELOPMENT

In his seminal paper first published in 1986, "No Silver Bullet: Essence and Accidents of Software Engineering," Frederick Brooks stated that software projects can become a "monster of missed schedules, blown budgets, and flawed products."[5] He then examined several technical developments that are most often advanced as a solution to the software development crisis and concluded that no single development will by itself improve productivity by as much as a factor of ten. Brooks did, however, predict that several of the emerging technologies (when used together) could be expected to yield the desired factor of ten improvement. In particular, Brooks listed object-oriented programming as having more promise than any of the other emerging technologies of his day.

There are several factors which are responsible for the software crisis, but they are all rooted in the fact that software is inherently complex. Grady Booch's classic book "Object-Oriented Design with Applications" gives four explanations for

Figure 1: THE THREE CORNERSTONES OF OOP



this unrestrained complexity: the problems software tries to solve are complex, it is very difficult to manage the software development process, software is very flexible, and it is difficult to characterize the behavior of the application.[6] Bar-David in an article titled "Object-Oriented Education and Training in the 1990's," asserts that programmers are responsible for the bottleneck in software production.[7]

A DESCRIPTION OF OBJECT-ORIENTED PROGRAMMING

Object-oriented programming has been referred to as a new programming paradigm, or way of viewing the programming process, and it is the programming methodology of choice in the 1990s.[8] In contrast to other programming paradigms such as the imperative-programming paradigm (languages such as C or Pascal) or the logic-programming paradigm (Prolog), OOP views a program as a collection of largely autonomous agents, called objects.[9] An object includes both data values and methods (similar to executing procedures). A programmer attempts to model or simulate desired behavior through the construction of objects. Objects exhibit three features which help the programmer take advantage of the object paradigm.

In an object, data and behavior are packaged or encapsulated. Encapsulation allows the programmer to hide internal details while providing a public interface to the object. This data hiding enhances reliability and modifiability of software by reducing the interdependencies between objects.

Another feature of an object, and perhaps its most powerful, is inheritance. Using inheritance, objects can acquire the attributes and behavior of other objects. This allows objects to share attributes and behaviors without separately duplicating the program code that implements them.

Finally, objects exhibit polymorphism. Polymorphism allows the shared code which objects acquire through inheritance to be tailored to fit the specific requirements of an object. This feature of an object allows for a higher level of abstraction in the design of software, since the programmer is only concerned about specifying actions and not how to implement these actions.

WHY OOP IS IMPORTANT

Object-oriented technology is not new, as its roots are in the early 70s when the SIMULA programming language became widely available. Like all emerging technologies, there is considerable discussion and dissension among "experts" as to the viability of object-oriented technology. Advocates like Ivar Jacobson point out that 5,000 programmers are currently developing systems based on object technology and conclude that the technology is proven and mature.[10] Others argue that IS will reject object-oriented programming and instead the technology will be hidden from application programmers under the cover of common IS tools.[11] A 1992 report by researchers at MIT's Sloan School of Management

asserts that object-oriented technology will be considered "experimental" for at least the near future.[12]

Object-oriented programming is attractive because it promises benefits in the areas of complexity control, reuse of standard components and increased ease of software maintenance. Each of these features should, in the long term, reduce the costs associated with software development.

Complexity Control

A standard technique of mastering a complex system is to decompose the system into smaller and smaller parts until each part becomes small enough to be understood. When the system is decomposed into modules, each carrying out a major step in the overall process, we are using top-down structured design. An alternate decomposition method is to decompose the system into objects, with each object having a unique behavior and modeling some object of the real system. This latter approach yields smaller systems than the top-down approach with the ability to incrementally grow from a reliable small system into a more complex system.[13]

Another method of dealing with complexity is through abstraction, which is the process of ignoring the nonessential details and dealing only with a generalized model of the object. Abstraction is easier and is a more normal part of the object-oriented development process than with other methods of software development.

Software Reuse

Since object-oriented techniques provide a mechanism for cleanly separating the interface of a module from its inner details, it is possible to routinely construct reusable software components. Brad Cox argues that object-oriented programming is a packaging technology, and he has introduced the term "Software-IC" to denote a reusable software component which is independent of the specific job at hand and is highly reusable in future jobs. He speculates that at some future date the software designer might be able to select from a catalog the software components needed to build a desired system.[14] The analogy with hardware design is obvious.

Software Maintenance

Systems built with an object-oriented approach are easier to modify than systems built with the classical top-down approach. There are several attributes of an object-oriented system which support this assertion. First, object-oriented systems are often smaller than equivalent implementations by other techniques. Next, object based systems can be modified over time rather than be abandoned or completely rewritten to adjust to changing requirements. Finally, since objects are encapsulated, it is easier to change the implementation of an object than it would be for the case of a conventional module.

Table 1: DPMA's IS'90 UNDERGRADUATE MODEL CURRICULUM

1.0 Computer Concepts:
- 1.3 Programming Languages and Applications Development Facilities
  - 1.3.6 OBJECT-ORIENTED LANGUAGES

3.0 Information Technology
- 3.1 Database
  - 3.1.2 LOGICAL DESIGN (DBMS INDEPENDENT DESIGN): ER, OBJECT- ORIENTED

4.0 Systems Theory and Development
- 4.3 Development Techniques
  - 4.3.3 OBJECT ORIENTED

OOP AND CURRENT CIS MODEL CURRICULA

The model curricula of both the Data Process Management Association (DPMA) and the Association for Computing Machinery/Institute of Electrical and Electronic Engineers-Computer Society (ACM/IEEE-CS) include object-oriented techniques. Additionally, both organization's models have been reviewed in order to find the emphasis and role these groups expect object-oriented techniques to play in the future training of undergraduates. The following is a brief review of each group's model and how object-orientation is included.

In DPMA's IS'90 Model Curriculum for a Four Year Undergraduate Degree[15], object-orientation appears in the areas of the Model Curriculum shown in Table 1.

The IS'90 model curriculum uses Bloom's Taxonomy, and "awareness", "literacy" and "concept" are the major requirements of object-orientation in the IS'90 curriculum. The exception is in the data base area where "Detailed Understanding" is required. This translates into "Be able to write syntactically correct...", "debug...", "implement... and maintain it", "apply principles of..to..", and "design a... for...", which is Level 3 of Bloom's Taxonomy (Application). The IS'90 sample courses do not include object-oriented topics until the third and fourth years. Furthermore, no course required an exit competency greater than Level 4, "Detailed Understanding".

Although the ACM/IEEE-CS Joint Curriculum Task Force (JCTF) "Computing Curricula 1991"[16] did not consider programs for information systems, they did define the term programming "... to denote the entire collection of activities that surround the description, development, and effective implementation of algorithmic solutions to well-specified problems." In addition, their report states that programming occurs in all subject areas. Their report has object-oriented topics interlaced with almost all major

subject areas. Specifically, these areas include: Database and Information Retrieval, Operating Systems, Programming Languages, Software Methodology and Engineering, Advance course electives on software design.

It is apparent that the JCTF believed object orientation would become an increasingly important methodology in the future. In their report several references are made to specific languages such as Smalltalk, SIMULA 67, Eiffel and C++. Furthermore, throughout the document an object-oriented approach to programming, analysis, and development is evident. The report makes reference to the contrast between object-oriented concepts and the more traditional methodologies and makes the recommendation that an object-oriented language should be used for the introductory course in Implementation G: A Program in Computer Science (Software Engineering Emphasis).

It appears that both the DPMA and ACM/IEEE-CS curriculum development groups expect up-to-date CIS curricula to begin to include coverage of object-oriented aspects of programming, analysis and design. The major subject areas within each model curricula reflect this shift to the object-oriented paradigm.

OOP AND CIS - THE EXPERIENCE AT PURDUE UNIVERSITY

Two years ago, Cain stated that although OOP technology has arrived, "CIS students cannot write OOP programs as part of their courses."[17] This statement was based on the presumption that COBOL was the primary language taught to CIS students. However, some educational institutions such as Dakota State University[18] and Miami University[19] are starting to incorporate OOP into their CIS curricula by using languages which contain provisions for OOP.

The Department of Computer Technology (CPT) at Purdue University has a nationally recognized undergraduate program in Computer Information Systems with special strengths in the areas of systems analysis and database. Recently the department began to update its curriculum with a long-term strategy that is based on two assumptions: 1) that programs will be produced by CASE tools, and; 2) the need for programmers will steadily diminish over the course of the current decade.

> ***The course is more than hand-waving about OOP — and students are given weekly assignments of simple programs which illustrate the various concepts and features of class development in C++.***

In recognition of current industry demand for programmers proficient in C++ and object-oriented programming, the CPT department has begun to offer an elective course titled An Introduction to Object-Oriented Programming Using C++. The course format consists of two 50-minute lectures and one 2-hour lab per week for a duration of 15 weeks. The course has a prerequisite of successful completion of a 3-credit course in the C programming language. Interest in the course is quite high, and the course has also been taken by non-CIS majors and graduate students.

Course Objectives

The course introduces the student to the concepts of object-oriented programming and explains how these concepts can be applied in C++. The course is more than hand-waving about OOP — and students are given weekly assignments of simple programs which illustrate the various concepts and features of class development in C++. In the last portion of the course, student teams enter, debug, document and modify a moderately sized (18 classes and 1700 lines of code) object-oriented system.

It is important to note that the primary emphasis of the course is learning about object-oriented programming. Only those C++ language features which are essential to OOP are covered. As pointed out by the class textbook author, "the programmer who successfully makes the paradigm shift, but does not know every last C++ feature, will be far ahead of the programmer who memorizes every C++ ampersand and keyword, but never learns the new approach to thinking about programming."[20]

Selecting the OOP Language

The concepts of object-oriented programming are far more subtle than those of structured programming and are concerned with the implementation of a certain view of what software should be. Because object-oriented concepts are so different from conventional procedure-based design techniques and programming languages like C, COBOL and FORTRAN, developing an object-oriented design is initially difficult for software designers. From a teaching and learning standpoint it would seem to be best to select a programming language which was designed specifically for OOP. If this premise is correct and there are no other considerations, then the language choice would be between Smalltalk and Eiffel.

Pure OOP Languages

Smalltalk

Smalltalk was first implemented in the 1970s at Xerox Corporation's Palo Alto Research Center (PARC) and represents everything as objects.[21] Smalltalk has a graphical user interface, and the system is especially good at rapid prototyping. However, Smalltalk produced systems have a reputation of having slow performance and requiring a lot of memory. Some companies are retraining their personnel by exposing them to object-oriented techniques through the use of the Smalltalk environment. By taking this approach programmers do not get bogged down in syntactic and semantic difficulties of (say) C++, and they can not revert to previously learned methods of programming. After sufficient training, the programmers continue to use object-oriented techniques, but they use another language (usually C++) to write programs which will be utilized in production.

Smalltalk appears to have reached critical mass by achieving an industry standard status.

Eiffel

Eiffel is a relatively new language whose goal is to support software engineering more effectively than C-based languages. [22] Eiffel attempts to build upon the worthy features of Ada, but without the complexity of Ada. The main disadvantage of Eiffel is that it is only available from one company, and this makes it difficult for Eiffel to achieve an industry standard status.

Hybrid Languages

Unfortunately, what is "best" in a technical sense is not necessarily best in an industrial sense. Educators must keep this tradeoff in mind as a language selection is made. In addition to the languages which were designed specifically for OOP, there are several languages which have been designed by starting with a procedures-oriented language and "grafting" object-oriented features to the original language. In this section three such languages are discussed. The first language, Object-oriented COBOL, appears to many to be an oxymoron and is included in this presentation solely because of COBOL's current industrial status. The second language, Objective-C, is derived from C and has achieved some degree of industrial adoption. The third language, C++, despite its disadvantages, has become an overwhelming success and is the standard for success against which all other object-oriented languages are judged.

COBOL

Object-oriented COBOL (OOCOBOL) is being developed by a task group called the Object-Oriented COBOL Task Group (OOCTG). Members of this task group have joined a technical committee (X3J4.1) of the American National Standards Institute (ANSI) COBOL Programming Language Committee X3J4.[23] The OOCTG has been meeting since 1989, and hopes to have a standard implemented by 1997. This group has been able to follow the development of object-oriented languages and can therefore take advantage of lessons learned, incorporating them into the OOCOBOL standards. OOCOBOL's potential major drawback is the obvious problem with legacy systems for which maintenance would be a nightmare using OOCOBOL to try to interface with them. However, "wrapping" them as objects may be a possible solution to this problem.[24]

COBOL is a long way from disappearing, but the OO techniques which promise a bright future for IS must be integrated with COBOL. It will be very interesting to see how this is done and when and if it will appear as truly OOCOBOL. In any case, this language will not be a serious contender for adoption by any organization until the language has full commercial support by one or more vendors. At the present time, language tutorials, user's guides, debugging tools, vendor support, and (most crucial of all) compilers simply do not exist.

Objective C

Objective-C was designed in the mid 80s with a goal of overcoming the performance deficiencies of Smalltalk.[25] This language uses Smalltalk concepts and syntax to add an object-oriented layer to C. In this way a user can utilize Smalltalk concepts while gaining C performance. Objective-C is currently provided by Stepstone,[26] NeXT and IBM. Few software libraries and tools are provided by third-party vendors.

C++

C++ appears to have an unassailable position as the language of choice for object-oriented programming by industry, and some have asserted that "the choice of C++ no longer remains a question of preference, but becomes a matter of necessity."[27] C++ is popular because its parent language C is firmly entrenched and under normal circumstances would not be at risk to be replaced in any case. C++ was designed to be a better C, support data abstraction, and to support object-oriented programming.[28] C++ compilers, libraries and tools are provided by major software companies and by a large number of third-party vendors. In addition to wholesale adoption by the industrial community, C++'s success seems additionally secure because an ANSI committee, X3J16, is already hard at work on standardizing the language.

However, it should be noted that C++ has its weaknesses and its critics. C++ has been designed to be compatible with C, and therefore compromises had to be made in the design of C++. The language is very complex, perhaps too complex. Unlike some other languages, the programmer is responsible for the details of dynamic memory management. One academician type referring to C++ has written "I am appalled at the monstrous messes that computer scientists can produce under the name of improvements."[29] The 1992 Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) conference proceedings contained the statement, "C++ disaster suffices to show how badly half-hearted solutions can fail." [30] Nevertheless, with the size of the C++ programming community continuing to double in less than a year,[31] C++ is perceived as the predominant object-oriented language of the 90s.

> ***...some have asserted that "the choice of C++ no longer remains a question of preference, but becomes a matter of necessity.***

Selecting the Programming Environment

In the past, entering program code, compiling, debugging and testing have occurred in a character-based programming environment. With the growing popularity of the graphic user interface (GUI), users are expecting their programs to run in a GUI-based environment, systems developers are using GUI-based computer aided software engineering (CASE) tools, and programmers too are starting to employ

tools which work in this modern environment.

For their OOP course using C++, the Purdue University CPT department selected Borland's Turbo C++.[32] Turbo C++ includes an integrated development environment (IDE) which allows the programmer to enter code, edit, compile, execute and debug completely within the IDE environment. The ability to do program development from one central environment increases productivity because it eliminates the necessity to switch between disparate environments.

Since most object-oriented programs consist of many files, the programmer is faced with the difficult task of keeping track of the dependencies between the various files. Borland's Project Manager "automates" dependency checking and makes it easier for the programmer to keep the executable file up-to-date.

Microsoft has recently introduced the Visual C++ development system, an upgrade to C/C++ 7.0, which one reviewer has termed "the new champ of Windows-based C++ compiler platforms."[33]. It is an integrated, Windows-hosted environment that combines the visually oriented techniques of Visual Basic with the flexibility of C++. With this system, an experienced programmer can write Windows applications in a fraction of the time it would take by using only the Windows software development kit (SDK).

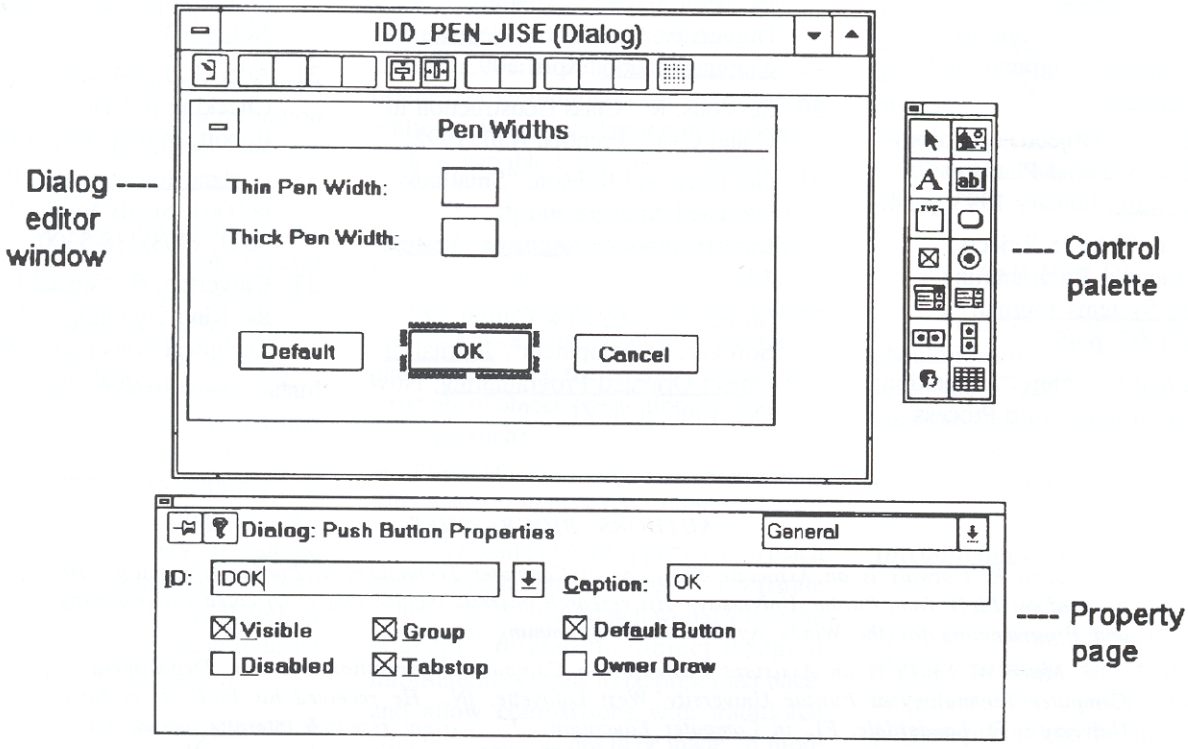
An example of Visual C++ programming is as follows. Suppose a Windows application needs a "dialog box" which the user will use to indicate the desired widths of a thin pen and a thick pen. The programmer uses Visual C++'s App Studio to easily and quickly generate the template code which is stored in the resource file.

Unfortunately, there are three topics which must be mastered in order to efficiently and effectively apply Visual C++ to Windows applications development. The developer must be proficient in C++, have a thorough understanding of the Microsoft Foundation Class (MFC) Library, and some knowledge of the Windows application programming interface (API).

CONCLUSION

Object-oriented programming is important because it offers improvements in complexity control, facilitates software reuse and promises to make software maintenance easier. As the CIS industry starts to adopt object-oriented techniques, it will actively recruit students who have received training in OOP. For this reason, universities should make plans for introducing an object-oriented programming course into their CIS curricula.

Figure 2: DESIGNING THE PEN WIDTHS DIALOG BOX WITH APP STUDIO



REFERENCES

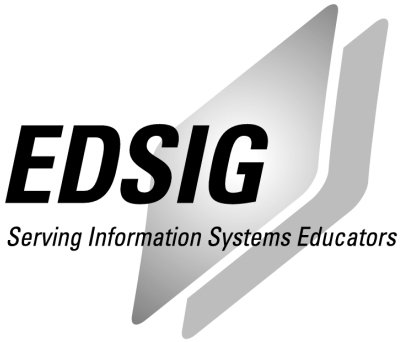
1. McGregor and Korson, Guest Editorial, Communications of the ACM, September 1990, p 38.
2. Garber, J. "Working Faster", Forbes, April 12, 1993, p 110.
3. Currid, C. "Plan now to prevent career dead ends for Cobol programmers", Infoworld, March 8, 1993, p 61.
4. Whitten, J. Computer Technology Dept., Purdue University, Private Communication, March 1993.
5. Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering", Computer, April 1987, p 10-19.
6. Booch, G. "Object-Oriented-Design with Applications", Benjamin/ Cummings, 1991.
7. Bar-David, T. "Object-Oriented Education and Training in the 1990s", Journal of Object-Oriented Programming, March/April 1993, p 24-30.
8. Pohl, I. "Object-Oriented Programming Using C++", Benjamin/Cummings, 1993.
9. Budd, T. "An Introduction to Object-Oriented Programming", Addison-Wesley, 1991.
10. Jacobson, I. "Is Object Technology Software's Industrial Platform?", IEEE Software, January 1993, p 24.
11. Fosdick, H. "Why IS Rejects Object-Oriented Programming", Enterprise Systems Journal, February 1993, p 45.
12. Fichman and Kemerer, "Adaptation of Software-Engineering Process Innovations: The Case of Object-Orientation", CISR WP No. 242, Center for Information Systems Research, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, June 1992.
13. Booch, G. ibid.
14. Cox, B. "Object-Oriented Programming: An Evolutionary Approach", Addison-Wesley, 1986.
15. DPMA, "IS'90 Model Curriculum for a Four Year Undergraduate Degree", DPMA, Park Ridge, IL, 1991.
16. ACM/IEEE-CS Joint Curriculum Task Force, "Computing Curricula 1991", ACM Press, 1991.
17. Cain, P. "Object-Oriented Programming and the CIS Curriculum", Journal of Information Systems Education, Vol. 3, No. 1, p 2-7.
18. White, B. Business and Information Systems, Dakota State University, Madison, SD, Private Communication, April 1993.
19. Rajkumar, T. Department of Decision Sciences, Miami University, Oxford, OH, Private Communication, April 1993.
20. Sessions, R. "Class Construction in C and C++", Prentice Hall, 1992.
21. Goldberg and Robson, "Smalltalk-90: The Language and its Implementation", Addison-Wesley, 1983.
22. Meyer, B. "The New Culture of Software Development", Journal of Object-Oriented Programming, Nov/ Dec 1990, p 76-81.
23. Schricker, D. "The Organizations Behind the Acronyms", Compilations, The Newsletter for the Micro Focus COBOL Community, March/April 1993, p 10.
24. Topper, A. "OOT and COBOL: How do they fit together?", Object Magazine, March-April, 1993, p 54-56.
25. Cox, B. ibid
26. "The Bottom Line: Using OOP in the Commercial Environment", OOPSLA 89 Workshop Report, Oct 1989.
27. Walker, G. "Why the Choice Must be C++", The C++ Journal, Vol. 2 No. 1, 1992.
28. Stroustrup, B. "The C++ Programming Language", Second Edition, Addison-Wesley, 1991.
29. Moody, R. "C In Education and Software Engineering", SIGCSE Bulletin, Vol. 23, No. 3, Sept. 1991, p. 45.
30. Meyer, B. "Ensuring Strong Typing in an Object-Oriented Language", OOPSLA '92 Conference Proceedings, ACM SIGPLAN Notices, Vol 27, No. 10, p 89.
31. Adcock, J. "Making a List and Checking it Twice", The C++ Report, March-April 1993, p 57.
32. Borland International, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408)438-5300.
33. Chiverton, B. "Visual C++ Enters the Ring Swinging and Scores a Technical Knockout", Microsoft Systems Journal, June 1993, p15.

AUTHORS' BIOGRAPHIES

John K. Gotwals is an Associate Professor of Computer Technology at Purdue University. He received his Ph.D. from Purdue University. His research interests include Object-Oriented Programming and Programming for the Win32 Application Programming Interface.

Mark W. Smith is an Assistant Professor in Computer Information Systems, Department of Computer Technology at Purdue University, West Lafayette, IN. He received his Ed.D. from Nova University, Ft. Lauderdale, FL, in Computer Education. His current research interests include OOP, computer ethics, computer anxiety, and micro-based COBOL program development. He is the co-author of two books on using Micro Focus COBOL and Micro Focus Personal COBOL.

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