View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AIS Electronic Library (AISeL)

Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2001 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2001

An Examination of Empirical Research in Object-Oriented Analysis and Design

Richard Johnson
Southwest Missouri State University

Follow this and additional works at: http://aisel.aisnet.org/amcis2001

Recommended Citation

Johnson, Richard, "An Examination of Empirical Research in Object-Oriented Analysis and Design" (2001). *AMCIS 2001 Proceedings*. 248.
http://aisel.aisnet.org/amcis2001/248

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2001 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

AN EXAMINATION OF EMPIRICAL RESEARCH IN OBJECT-ORIENTED ANALYSIS AND DESIGN

Richard A. Johnson
Southwest Missouri State University
richardjohnson@smsu.edu

Abstract

Object-oriented systems development (OOSD) is viewed by many as the best available solution to the ongoing "software crisis." However, some caution that OOSD may be so complex that it will never become a mainstream methodology. Of particular importance to successful OOSD is object-oriented analysis and design (OOAD), the cornerstone of any serious OO project. This paper reviews a wide range of empirical studies on OOAD involving human subjects over the past decade. While OOSD is a vitally important approach to modern systems development, it is not without its difficulties, as evidenced by often conflicting results within the exercise of OOAD.

Introduction

Object and component technologies, rapidly maturing branches of information technology, are becoming pervasive elements of systems development, especially the recently popular Internet applications. However, mainstream object-oriented systems development (OOSD), consisting of object-oriented analysis and design (OOAD) and object-oriented programming (OOP), has a history of difficulties (Pancake, 1995) and is still struggling to gain widespread acceptance (Hapgood, 2000). Some believe that "technology adoption is mostly the result of marketing forces, not scientific evidence" (Briand et al., 1999, p. 388) and, as Smith and McKeen (1996) have observed, object technology is "still long on hype and short on results . . ." (p. 28). The gurus of OOSD (Booch, 1994; Coad and Yourdon, 1991; Coleman et al., 1994; Jacobson et al., 1995; Rumbaugh et al., 1991) continue to tout its vast superiority over conventional systems development, even to the extent of developing a "unified software development process" (Jacobson et al., 1999).

The advocates of OOSD claim many advantages including easier modeling, increased code reuse, higher system quality, and easier maintenance. However, some express serious concern about certain disadvantages of OOSD, such as its difficulty to learn, slower development time, and poorer run-time performance (Fichman and Kemerer, 1993). It is well understood that analysis and design are extremely critical aspects of successful systems development (Partridge 1994, Brooks 1987), especially in the case of OOSD. For example, analysis and design account for about 70% of the effort in developing OO systems, but only about 50% of the effort in developing conventional systems (Meyer 1988). As the development of any successful information system must begin with a well-conceived and implemented analysis and design, this study will focus on the most recent empirical evidence on the pros and cons of OOAD.

Background

What Is OOAD?

The development of object-oriented systems became possible with the proliferation of object-based and object-oriented programming languages in the early 1980s. As is often the case, programming languages are developed long before the theory of how to use them effectively and efficiently (Tkach and Puttick 1994). While small systems may be developed successfully without the aid of a formal system of analysis and design, larger, "industrial strength" (Booch 1994) projects require a more systematic approach (Partridge 1994).

OOD methods emerged in the mid-1980s and OOA methods in the late 1980s. Notice the backward migration from OOP to OOD and from OOD to OOA. An OOAD methodology consists of processes (methods describing "how to"), techniques (formalisms, models, notation), and, possibly, tools (CASE tools to create models and enforce relationships) (Monarchi, Booch, Henderson-Sellers, Jacobson, Mellor, Rumbaugh and Wirfs-Brock 1994). Some of the more significant published methods of OOAD reviewed in the literature include those of Booch (1994), Wirfs-Brock et al. (1990), Coad and Yourdon (1991), Rumbaugh et al. (1991), Shlaer and Mellor (1992), Jacobson et al. (1992), and Coleman et al. (1994). In fact, the number of OOAD methods exploded from fewer than 10 to more than 50 between 1989 and 1994. The field of OOAD has made particularly important strides in just the past few years with the development of the unified modeling language (UML), the current standard graphical language for OO analysis and design. UML started as a unification of the Booch and OMT methods at Rational Corporation in 1994 and incorporated OOSE by 1996. The Object Management Group (OMG) accepted UML as a standard modeling language in November 1997 after widespread contribution from industry (Booch, 1999).

OOA is the process of converting the real-world problem into a model using objects and classes as the modeling constructs (Booch 1994). The objects identified from OOA are called semantic objects since they have meaning in the problem domain. An OOA model should ideally be understandable by application experts who are not programmers. OOD is the process of converting the problem model (from OOA) into a solution model based on objects. During OOD, new objects, not found in the OOA models, are added for implementation purposes. The implementation details of semantic objects are also added (short of writing actual code in the target OOPL). OOD is usually viewed as adding more detail to OOA with a special focus on modeling the implementation of the proposed system. OOD can be executed at different levels such as class design, system design, and program design (de Champeaux 1992). According to Booch (1994), OOD "encompasses the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design" (p. 39). Thus, OOD produces models of the proposed information system (the solution) rather than models of the real-world system (the problem).

Empirical Studies in OOAD

Early Studies on OOAD

Many early studies of OOAD (1992-1996) made direct comparison between OO and conventional methods. Boehm-Davis and Ross (1992) compared the quality of designs and solutions for various projects using three different types of systems development methodologies: procedural, data-oriented (Jackson System Development, or JSD), and object-oriented. The eighteen subjects were professional programmers divided into three groups of six, each group having received training and/or possessing previous experience in one of the three different methodologies. The subjects were asked to provide designs and write pseudo-code for three different systems. Data were collected on solution completeness, time to design and code, and solution complexity. The findings reveal that the JSD and OO groups generated significantly more complete solutions, required significantly less development time, and produced less complex solutions than the procedural group. The accomplishments of the OO group look even more impressive when noting that the JSD group had three to four times more overall development experience and more than twice the experience with its respective methodology compared to either the procedural or OO groups. Thus, one can conclude that for professional developers, OO designs and solutions are of higher quality and take less time than procedural designs and solutions. While the results look somewhat encouraging for OO, the experimental design of the research has some serious limitations, primarily with the inability to control for the level of prior experience within the respective methodologies.

Vessey and Conger (1994) also compared the same three types of analysis methods: process-oriented (structured), data-oriented (Jackson System Development), and object-oriented (Booch). A total of six software engineering students, inexperienced in any analysis method, received the same training in all three methods during a university course. They were then assigned to one of three groups (two students per group) and given equally complex analysis problems to solve using one of the three analysis methods. The researchers performed a protocol analysis and determined that novice analysts found OOA more difficult to learn and apply than data-oriented analysis and data-oriented analysis more difficult to learn and apply than process-oriented analysis. This study seemingly contradicts the finding of Boehm-Davis and Ross (1992) that OO is easier to apply. While the methods used by developers in the Vessey and Conger (1994) study were almost identical to those in the Boehm-Davis and Ross (1992) study, the former study used students instead of experienced developers and used a much smaller sample size (n=6 vs. n=18). Also, the students were not randomly assigned to groups.

Pennington et al. (1995) performed a protocol analysis on a total of ten experienced, professional developers. Three were expert procedural developers, four were expert OO developers, and three were novice OO developers (who were, however, expert procedural developers). All three groups were given a relatively simple swim meet scoring problem and asked to create a complete design using their respective methods. Completed designs were judged in terms of quality while developers were evaluated on productivity. The results revealed that the designs of the OO experts were more complete but took more time compared to the procedural experts. Even though they took more time, the OO experts were graded more efficient than the procedural experts when overall design quality was considered. The study concludes that OO designs are of higher quality than procedural designs and take less time to complete.

Hardgrave and Dalal (1995) performed a lab study of 56 advanced undergraduate MIS majors, all enrolled in a senior-level DBMS course, to compare two competing data modeling techniques: the extended entity-relationship (EER) model (McFadden and Hoffer, 1991) and the Object Modeling Technique (OMT) of Rumbaugh et al. (1991). The independent variables were modeling technique (OMT or EER) and complexity of the resulting model. The students were randomly assigned to one of four groups, with each group given a previously prepared, completed model to review (simple OMT, complex OMT, simple EER, and complex EER). The students in each group, who had already received training in the techniques, were provided with two additional one-hour lectures specifically on their respective models. They were then asked to take a test on their understanding of the models and complete a follow-up questionnaire. The dependent variables were level of understanding (measured by the score on the test), time to understand (measured by the time to complete the test), and perceived ease-of-use (measured by item scores on a questionnaire). The results indicated that, for both simple and complex systems, OMT models were more quickly understood than EER models. However, no significant difference was found for the depth of understanding and the perceived ease of use of the two methods, regardless of task complexity. Thus, OO modeling techniques may be more quickly understood, but not more completely understood, compared to data-oriented techniques. One possible shortcoming of this study is that it compares object-oriented to data-oriented modeling techniques. These two methods are much more closely related than object-oriented and process-oriented techniques, so differences in understanding or perceived ease of use may be difficult to detect, and even if detected, less relevant to the concerns of many practitioners and researchers.

Wang (1996a) performed an experiment using thirty-two undergraduate students with no previous systems analysis training or experience. The subjects were randomly divided into two groups. One group was trained for five hours on the data flow diagram (DFD) method, while the other group was trained for five hours on an object-oriented analysis method. The subjects were then presented with a mini-case in management information systems analysis. The OO group spent significantly less time on their analyses of the problem and created solutions that were significantly more accurate. After completing the analysis, the subjects responded to a questionnaire concerning their perceptions of the analysis method used. The OO group reported that the OOA method was easier to learn and understand. The OOA method was also rated superior overall. This study confirms the results of several previously cited studies: OOA produces higher quality models more quickly than procedural analysis.

In a separate study, Wang (1996b) again compared a structured method of analysis (DFD) with object-oriented analysis (OOA) using two groups of inexperienced undergraduate MIS majors. Students were randomly assigned to two groups, 24 in the DFD group and 20 in the OOA group. Each participant learned his respective analysis method and created analysis diagrams based on information in a mini-case study. The total time allowed for training and problem solving was 7.5 hours spanning several class sessions. The two dependent variables were the syntactic and semantic accuracy (in conveying system requirements) of the resulting analysis diagrams. Using ANOVA techniques, the results indicated that the syntactic accuracy for the DFD group was significantly greater in the early sessions, but that syntactic accuracy for the OOA group was significantly greater in the last session. However, there was no significant difference in semantic accuracy for the DFD and the OOA groups. Apparently contradicting the results of this researcher's previous study (Wang 1996a), this experiment concludes that OOA appears more difficult to learn than DFD, and that OOA does not produce solutions of higher quality.

Another important benefit claimed for OOAD is improved communication among development team members, as well as between users and developers (Garceau et al., 1993). The assumption is that OO is easier to understand, but it is not clear whether this should lead to increased or decreased communication. No empirical research was found on improved communication between users and developers, but one study focused on developer interaction during the design phase of OO projects (Herbsleb et al., 1995). In this research, several field studies were conducted using developers' time sheets, videotapes of meetings on design activities, and semi-structured interviews with developers. Results indicated that when OOD methods are used, fewer spontaneous episodes of clarification occur. Also, planned summaries and walk-throughs occur much more often when using OOD. More attention was given to the reasons for specific design choices for the OO projects. OOD seems to encourage a deeper inquiry into the reasons underlying design decisions but less inquiry into the requirements. The authors believe these findings indicate improved communication in software development teams, which leads to greater understanding of requirements. However, there may be alternative explanations. For example, fewer spontaneous episodes of clarification could occur if developers wish to disguise a lack of understanding. The increased number of planned summaries and walk-throughs could result if developers perceive a lack of understanding among peers. Thus, the study may indicate that OOD decreases one form of communication and increases another simply because it is new or more difficult to understand, not because it is easier or more natural.

Supporters of OOAD claim that thinking in terms of interacting objects, rather than in terms of functions or procedures, should be more natural to humans (Pancake, 1995). Davies et al. (1995) set out to test the claim that OO decomposition of the problem domain is more natural to the ways of human cognition than functional decomposition. Twelve expert and twelve novice programmers were presented with cards containing fragments of code from a large C++ library for graphics applications. Their task was to sort the cards according to any criteria they felt appropriate. The purpose of the sort was to determine whether the subjects would perform functional or object-oriented decompositions of the problem domain. Subjects performed the sorting of code fragments and reported the reasons for their sorting (categorized as either function-based or object-based). The results showed that expert subjects seemed to focus more on the functional properties of the code while the novice subjects tended to classify the code fragments according to important features of the OO paradigm (class membership, object similarity, or inheritance relations). According to the authors, the "results appear to suggest fairly clearly that functional information is of much greater importance to experts than is information about objects and their relations" (p. 242). The implication is that OO decomposition is not more natural for expert developers, as was expected by the researchers. Of course, an alternative explanation is that experts are simply more experienced with functional decomposition and tended to see the code fragments in that way.

Agarwal et al. (1996) performed a thorough experiment comparing the ability of novice analysts to correctly perform a requirements analysis using either a process-oriented (PO) or an object-oriented (OO) analysis methodology. A total of 43 undergraduate students (with no prior training or experience in any type of systems analysis) were randomly divided into two groups: a PO group (n=24) and an OO group (n=19). Each group was trained six hours in its respective analysis methodology—the DeMarco (1978) method for the PO group and the Coad and Yourdon (1990) method for the OO group. Individuals in each group were then presented with two problems to analyze—one problem was clearly more function-strong (PO) while the other was more structure-strong (OO). According to the theory of cognitive fit, the PO group should perform better on the PO problem, while the OO group should perform better on the OO problem. The researchers found that the PO group had significantly better overall performance than the OO group on the PO task, but that there was no difference in overall performance between the two groups on the OO task. The researchers concluded that PO methodologies should be easier for novices to learn than OO methodologies, possibly because people may have a greater tendency to reason procedurally.

More Recent Studies on OOAD

During the past five years (1996-2001), empirical studies of OOAD have shifted their focus from direct comparisons of OO and conventional methods to an exploration of the characteristics of OOAD that contribute to the quality of completed OO systems. This shift is likely due to the increased overall acceptance of OOSD, leading researchers away from comparisons to traditional methods.

Briand, et. al (2000) discovered that the frequency of method invocations and the depth of inheritance hierarchies are the major determinants of fault-proneness of resulting software classes. Eight three-person teams of upper division undergraduate students, with no previous OO experience, were taught OOAD. Each team developed a medium-sized MIS for a hypothetical video rental business. The OMT analysis and design method (Rumbaugh et al.) was used with C++ as the implementation language. Independent testers, consisting of experienced software professionals, evaluated the coded classes for faults. Existing measures of coupling (classes using methods or attributes in other classes), cohesion (methods within a classes using common attributes of the class) and inheritance (classes deriving methods from ancestor classes) defined at the class level were used as independent variables to predict the probability of fault-proneness in class code (the classes investigated were either developed from scratch or were extensive modifications of library classes). Univariate analysis revealed that increased levels of coupling and inheritance have a significant impact on fault-proneness of classes while cohesion does not. Multivariate analysis showed that models involving coupling and inheritance measures could be developed to automatically detect faulty classes with an accuracy rate approaching 90%.

A similar study by El Emam, et al. (2001) focused only on those metrics that are available at the design stage. The measures involved two characteristics of OO design classes, coupling and inheritance (briefly explained above). The applications involved in the study were two consecutive releases of a commercial word processing program written in Java. Data were collected on faults reported by users of both versions so that classes could be identified as either faulty or not. Design metrics were applied to all classes in both versions to find the relationships between measures of coupling and inheritance and fault-proneness of the classes. A multivariate model including class size (number of attributes and methods), export coupling (number of times a class's methods are used by other classes), and inheritance depth (the number of levels of inheritance for a class) measures resulted in an R-squared value of .24, with export coupling having the predominant influence on fault-proneness.

A study by Laitenberger et al. (2000) took a different approach to investigating characteristics of OO designs, specifically design documents utilizing the Unified Modeling Language (UML). The independent variable in this study is the type of reading technique used by individuals to detect defects in UML design documents for OO systems. The idea is for knowledgeable individuals to read design documents to detect defects prior to implementation of the designs. Eighteen students (many of who were experienced practitioners) were used to examine the design documentation for two simple systems, a Web-based quiz system and a point of sales system. The two reading techniques investigated in this study are checklist-based reading (CBR) and perspective-based reading (PBR). The CBR approach focuses on a defined set of questions addressing both "Where to look" and "How to detect" problems. On the other hand, PBR is a scenario-based technique that goes beyond a fixed set of questions to provide guidance to the inspectors on how to proceed based on the perspectives of the stakeholders of the system. Results indicate that PBR is much more effective and efficient for UML documents of OO systems than CBR. This study contrasts a manual, human approach to defect detection at an early stage of design to an automated approach using metrics at a late stage of design.

Discussion

General Conclusions About OOAD

A total of twelve empirical studies, representing some of the best available in the field of OOAD, have been presented. In nearly every instance where studies were favorable to OOAD, higher quality and productivity were cited as primary benefits. On the other hand, nearly every negative result focused on the difficulty of learning OOAD or the inherent complexity of OO designs. These results are consistent with the anecdotal OO literature. In any event, the results suggest that while OOAD may be somewhat more difficult to learn than conventional methods, the effort spent in education and training may ultimately pay off in increased quality and productivity.

Some studies discussed above present mixed results on other important OOAD issues. For example, the OO paradigm was found to be more natural for developers (Davies et al., 1995), although the logical derivation of this conclusion from the data is highly suspect. The conclusion that OOAD enhances communication (Herbsleb et al., 1995) may actually highlight a potential disadvantage of OOAD, i.e., that OOSD may be more confusing, thus causing an increased level of communication.

Nearly all studies where only negative results were obtained stemmed from the use of inexperienced students as subjects. This suggests that learning can play a tremendous role in the effectiveness of OOAD. Students given only a few hours or weeks of training in OOAD should not be expected to perform OO tasks particularly well, especially given that OOAD may be somewhat difficult to learn. The conventional wisdom is that proficiency in OOAD may require six to eighteen months of full-time experience (Fayad et al., 1996). Thus, many of the negative results could be attributed to the types of subjects chosen and the amount of training provided.

Methodological Issues

The results of this review also point to several possible methodological weaknesses in empirical studies of OOAD. First, many empirical studies in OOAD use a small sample size and otherwise poor experimental design. Sample sizes of one or two per treatment are very susceptible to validity problems. Also, studies may not be able to detect significant differences between procedural and OO methods due to a lack of statistical power. Additionally, poor experimental designs that fail to randomly assign subjects to treatments or otherwise fail to control for developer experience call results into question.

As discussed earlier, studies often use inexperienced students as subjects. Such practices may be acceptable when the purpose of the research is to explore the difficulty of learning OOAD, but not when research questions focus on the quality and productivity of models or completed systems. Also, the question of learning OOAD may be even more critical to experienced procedural developers who may be forced by management to make the transition to OO, but no studies were found that specifically address this group.

Another potential problem exists with studies that attempt to quickly train novice students in OOAD. Instructors at universities where such studies are conducted are likely to be significantly less experienced in the new OO methodologies than the more established procedural methodologies. This condition could result in less than optimum conditions for effectively and efficiently transferring complex OO knowledge, making it even more difficult for students to adequately learn OO.

Future Research

Clearly, more research of higher quality is needed to determine with greater certainty how OOAD compares to conventional methods. Laboratory experiments could be designed to determine how well subjects, especially students, are able to learn and apply OOAD. Field studies and survey research on experienced developers could explore the transition to OOAD in industry and the effectiveness of OOAD on complex projects. Researchers could also investigate whether learning OOAD is more difficult for novice or experienced developers. Longitudinal studies should be conducted to determine if those who found OOAD difficult to learn eventually mastered the techniques and whether those who found OOAD less difficult to learn were any more successful at applying the methodology.

One problem with conducting future research on OOAD involves clearly defining a strategy to address specific research questions. The following list presents several dimensions that empirical researchers should consider in the design of future experiments or field studies on the pros and cons of OOAD:

- Types of methodologies to be compared: process-oriented, data-oriented, object-oriented
- Types of applications to be developed: function-intensive, data-intensive, hybrid
- Complexity of applications to be developed: simple classroom vs. complex industrial
- Level of previous OO development experience: novice vs. experienced
- Type of previous experience: process-oriented, data-oriented, object-oriented
- Type of experiment: laboratory vs. field (including survey research)
- Sample size: small vs. large
- Time frame of research: cross-sectional vs. longitudinal

As is apparent from the list above, the choices for empirical investigation of OOSD are many. An ideal situation would be to collect detailed data on experienced individual developers or development teams who create identical complete real-world systems (perhaps of varying complexity) using both conventional and OO methods.

Regardless of the particular research question involved, better experimental designs with tighter controls and larger samples could enhance validity. The obvious dilemma in this type of research is obtaining the cooperation of sufficiently large numbers of qualified subjects for laboratory or field studies. However, without adequate experimental designs, a quick resolution to the OO controversy will remain elusive.

References

Agarwal, R, Sinha, A.P., and Tanniru, M. (1996). Cognitive fit in requirements modeling: a study of object and process methodologies. *Journal of Management Information Systems*, 13:2 (Fall), 137-162.

Boehm-Davis, D. and Ross, L. (1992). Program design methodologies and the software development process. *International Journal of Man-machine Studies*, 36, 1-19.

Booch, G. (1994). *Object-oriented analysis and design with applications*, 2nd ed. Benjamin/Cummings (Redwood City, CA).

Briand, L., Arisholm, E., Counsell, S., Houdek, F., and Thevenod-Fosse, P. (1999). Empirical studies of object-oriented artifacts, methods, and processes: state of the art and future directions. *Empirical Software Engineering: An International Journal*, 4:4 (December), 387-404.

Briand, L., Wust, J., Daly, J., and Porter, D. (2000). Exploring relationships between design measures and software quality in object-oriented systems. *The Journal of Systems and Software*, 51, 245-273.

Brooks, F.P. (1987). No silver bullet: essence and accidents of software engineering. *IEEE Computer*, 20:4 (April), 10-19.

Coad, P. and Yourdon, E. (1991). *Object-oriented analysis*, 2nd ed. Yourdon Press (Englewood Cliffs, NJ).

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P. (1994). *Object-oriented development: the fusion method*. Prentice-Hall (Englewood Cliffs, NJ).

Davies, S.P., Gilmore, D.J. and Green, T.R.G. (1995). Are objects that important? Effects of expertise and familiarity on classification of object-oriented code. *Human-Computer Interaction*, 10, 227-248.

DeMarco, T. (1978). *Structured analysis and system specification*. Prentice-Hall (Englewood Cliffs, NJ).

El Emam, K., Melo, W., Machado, J. (2001). The prediction of faulty classes using object-oriented design metrics. *The Journal of Systems and Software*, 56, 63-75.

Fayad, M., Tsai, W. and Fulghum, M. (1996). Transition to object-oriented software development. *Communications of the ACM*, 39:2 (February), 108-121.

Fichman, R.G. and Kemerer, C.F. (1992). Object-oriented and conventional analysis and design methodologies: comparison and critique. *IEEE Computer*, 25:10 (October), 22-39.

Fichman, R.G. and Kemerer, C.F. (1993). Adoption of software engineering process innovations: the case of object orientation. *Sloan Management Review*, 34:2 (Winter), 7-22.

Garceau, L., Jancura, E., and Kneiss, J. (1993). Object-oriented analysis and design: a new approach to systems development. *Journal of systems management*, 44:1 (January), 25-33.

Hardgrave, B. and Dalal, N. (1995). Comparing object-oriented and extended-entity-relationship data models. *Journal of Database Management*, 6:3 (Summer), 15-21.

Herbsleb, J., Klein, H., Olson, G., Brunner, H., Olson, J., and Harding, J. (1995). Object-oriented analysis and design in software project teams. *Human-Computer Interaction*, 10, 249-292.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The unified software development process,* Reading, MA: Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. (1995). *Object-oriented software engineering: a use case driven approach*, 2nd ed. Addison-Wesley (Wokingham, England).

Laitenberger, O., Atkinson, C., Schlich, M., and El Emam, K.. (2000). An experimental comparison of reading techniques for defect detection in UML design documents. *The Journal of Systems and Software*, 53, 183-204.

McFadden, F. and Hoffer, J. (1991). *Database Management*, 3rd ed. Benjamin/Cummings (Redwood City, CA).

Pancake, C.M. (1995). The promise and the cost of object technology: a five-year forecast. *Communications of the ACM*, 38:10 (October), 33-49.

Pennington, N., Lee, A.Y., and Rehder, B. (1995). Cognitive *activities and levels of abstraction in procedural and object-oriented design. Human*-Computer Interaction, 10, 171-226.

Rumbaugh, J., Booch, G., and Jacobson, I. (1998). *The unified modeling language reference manual,* Reading, MA: Addison-Wesley.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991*). Object-oriented modeling and design*. Prentice Hall (Englewood Cliffs, NJ).

Shlaer, S. and Mellor, S. (1989). *Object lifecycles: modeling the world in states*. Prentice-Hall (Englewood Cliffs, NJ).

Smith, H.A. and McKeen, J.D. (1996). Object-oriented technology: getting beyond the hype. *The DATA BASE for Advances in Information Systems*, 27:2 (Spring), 20-29.

Vessey, I. and Conger, S. (1994). Requirements specification: Learning object, process, and data methodologies. *Communications of the ACM*, 37:5 (May), 102-113.

Wang, S. (1996a). Toward formalized object-oriented management information system analysis. *Journal of Management Information Systems*, 12:4 (Spring), 117-141.

Wang, S. (1996b). Two MIS analysis methods: an experimental comparison. *Journal of Education for Business*, 71:3 (Jan/Feb), 136-142.

Wirfs-Brock, R. and Johnson, R. (1990). Surveying current research in object-oriented design. *Communications of the ACM*, 33:9 (Sep.), 105-124.