**Communications of the Association for Information Systems**

Volume 16 Article 43

12-8-2005

# Special Theme of Research in Information Systems Analysis and Design -III Teaching Systems Analysis and Design: A Case for the Object Oriented Approach

Sridhar P. Nerur
*University of Texas at Arlington*, snerur@uta.edu

Craig W. Slinkman
*University of Texas at Arlington*, slinkman@uta.edu

RadhaKanta Mahapatra
*University of Texas at Arlington*

Communications of the Association for Information Systems

# SPECIAL THEME OF RESEARCH IN INFORMATION SYSTEMS ANALYSIS AND DESIGN - III

# TEACHING SYSTEMS ANALYSIS AND DESIGN – A CASE FOR THE OBJECT ORIENTED APPROACH

RadhaKanta Mahapatra
Sridhar P. Nerur
Craig W. Slinkman
*Department of Information Systems and Operations Management*
University of Texas at Arlington
mahapatra@uta.edu

**ABSTRACT**

Object oriented technologies are widely accepted in software development. A survey of universities run in 2005 found that most schools recognize the need to teach OO languages. However, they continue to teach structured analysis and design. In this article we argue that this approach is a fundamental conceptual mismatch. Further, we contend that a pure OO curriculum involving OO languages and OO analysis and design is advisable in our efforts to equip our students with the knowledge to be successful as software developers. We offer ways to transition to a curriculum that emphasizes the OO philosophy of development.

**Keywords:** object oriented systems analysis and design, software development, OO languages, OO curriculum, structured analysis and design

## I. INTRODUCTION

Information technology (IT) is advancing rapidly. As a consequence, Information Systems (IS) professionals are under constant pressure to upgrade their skills and keep abreast of new technologies [Lee et al, 1995]. IS, as an academic discipline, is mandated to ensure a steady supply of graduates to meet the economy's demand for IS professionals. The rapid obsolescence of IT skills makes the IS academic's job challenging. Departments need to monitor the advances in IT continually and update the curriculum appropriately to maintain its relevance. Such curricular changes may take one of two forms.

1. Adding new courses to the curriculum to satisfy the demand for new knowledge and skill. For example, to meet the increasing demand for information security specialists, several schools started offering security courses.

2. Updating the contents of existing courses as their contents become obsolete. For example, in response to increasing demand for JAVA programmers many schools replaced COBOL with JAVA as the language of choice in programming language

courses. This kind of change involves modifying course content and is somewhat trickier to accomplish. The primary reason for this difficulty is that, in the business world, technologies do not become obsolete overnight. Two competing technologies often continue to coexist for an extended time. During this time of transition, academics often face the dilemma of when to update the course content. Their decision impacts how successfully they will fulfill their manpower mandate. Too much delay in updating the course content is likely to create a shortage of professionals with the right skills.

The IS2002 model curriculum [Gorgone et al., 2003] created by the academic community is helpful in providing guidance about the structure of a curriculum. However, as stated under its guiding principles, providing detailed prescriptions for individual courses is outside its scope [Gorgone et al, 2003, pp. 5]. Academic IS journals such as the *Communications of the AIS* provide an excellent forum for discussing such issues. In the early 2000's, Computer Science, which also deals with issues related to technology obsolescence, engaged in an extended debate on how to integrate Object-Oriented (OO) technologies into first level courses (example, [Marion, 1999; Mitchell 2001]). The objective of this paper is to initiate a similar discussion on what should be taught in the Systems Analysis and Design course.

The ability to analyze, design, build and maintain information systems forms a critical skill set of IS graduates [Couger et al, 1995; Gorgone et al, 2003]. Therefore, programming, and analysis and design courses form part of the core requirement of all undergraduate IS curricula.

Systems development underwent major changes since the 1990's, primarily because of the rapid adoption of object-oriented technologies. A survey of IS project managers conducted in 2001 by Schambach and Walstrom [2002] found that OO analysis and design (OOAD) methods were more useful and were more frequently used compared to structured analysis and design methods. This study also predicted an increasing trend in the use of OOAD methods.

A search of jobs listed in Monster.com in February 2005 found that JAVA-related jobs outnumbered COBOL related jobs by a ratio of 10 to 1 (Table 1). This data substantiates that knowledge and skills related to OO development are in much higher demand compared to those needed for procedural programming and structured development. An examination of current popular books and extant literature on software engineering further indicates the importance given to OO and modeling topics such as design patterns and the Unified Modeling Language [Larman 2005].

Table 1. Search Results of Jobs listed in Monster.com,Feb 24, 2005 at 3:00 p.m.

| Keyword | Last 24 hour postings | Last 3 day posting | Last 7 day posting |
|---|---|---|---|
| Java | 541 | More than 1000 | More than 1000 |
| Cobol | 50 | 108 | 180 |

## THE STATE OF PEDAGOGY

We surveyed IS academics in AACSB accredited business schools to assess the state of pedagogy related to software development. The details of this and an earlier survey are presented in the following section. We found that all respondents to our 2005 survey indicated that they offer one or more courses on OO programming languages, but less than a quarter offer courses on OOAD. Thus, while most schools have upgraded their programming course contents to include OO concepts, many have not done the same in their analysis and design courses. We, as an academic community, need to take a closer look at what we teach in our analysis and design courses.

The objective of this paper is to initiate a discussion on this issue. Specifically, we want to make a case for teaching object-oriented analysis and design.

## ORGANIZATION OF THIS PAPER

The remainder of the article is divided into four sections. In Section II we describe the two surveys and discuss their results. In Section III we make the case for why OOAD should be taught rather than structured analysis and design. Section IV presents our suggestions on how to make the transition to teaching OOAD. Finally, the research is summarized and conclusions are drawn in Section V.

## II. THE CURRENT STATE

We conducted two surveys of IS people in academia, one in 2002 and the other in 2005, to assess the state of IS pedagogy in software development. We are interested in understanding the state of IS pedagogy because we are engaged in teaching courses on programming, and analysis and design. We have about 40 years of combined experience in teaching these courses. We also conduct research on software development methodologies. We were aware that IS practitioners were making the transition to OO technologies. Our goal was to survey our peers in other universities to obtain guidance in updating our own curriculum.

### SURVEY METHODOLOGY

In November 2002, we randomly selected 100 universities from the list of AACSB accredited schools located in the USA. We visited the websites of the universities selected to identify those that offer undergraduate business degrees with an IS major. A survey (shown in Appendix I) was e-mailed to a contact person (mostly the department chair) in the academic unit offering the IS degree. We followed up with a reminder message to those that did not respond to our first request. Of the 83 surveys initially sent we received a total of 47 usable responses, a response rate of 57%. We conducted a second survey in January 2005 using an updated list of participants in our first survey. This time we added C# to the programming language list in the survey instrument. The second survey was emailed to 84 departments and we received 31 usable responses, a 37% response rate.

### PROGRAMMING

Figure 1 and Table 2 show the percentages of departments offering programming courses in one or more of the languages listed in our survey instruments. The data clearly shows that VB and JAVA were the two most popular languages in 2002, with each being offered by more than 60% of the respondents. More than a quarter of schools at this time offered courses in COBOL, and approximately 20% taught programming in C. The programming languages offered changed somewhat by early 2005. JAVA and VB.Net were the two most popular offerings. None of the respondents offered courses in C programming. COBOL is also in lower demand with 11% offering COBOL courses in 2005 compared to 26% in 2002. The change in the coverage of OO programming is evident. OO languages listed in our 2005 survey were C++, JAVA, VB.Net, and C#. In 2002 19% of the respondents did not offer a course in OO programming. In contrast, all respondents to our 2005 survey offer at least one course on OO programming. As discussed in Section III, this universal coverage of OO programming in IS pedagogy offers important implications for teaching OOAD.

Table 2. Percentage of Schools Offering Programming Languages

| LANGUAGE | 2002 (n=47) | 2005 (n=27)[1] |
|---|---|---|
| COBOL | 26 | 11 |
| C | 19 | 0 |
| VB | 64 | 22 |
| C++ | 32 | 15 |
| JAVA | 66 | 59 |
| VB.NET | 34 | 63 |
| C# | | 15 |

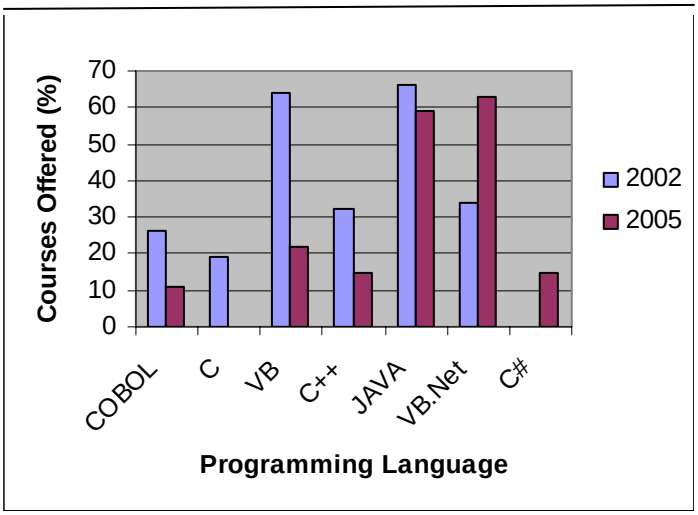


Figure 1. Programming Languages Offered

## ANALYSIS AND DESIGN METHODOLOGY

Based on informal interactions with our peers in other academic institutions, we were aware that some schools cover either structured or OO methodology, whereas others provide extensive coverage of one methodology while offering a brief introduction to the other. Question 2 of our survey was designed to gather information about the methodology focus of the analysis and design course. The responses to this question are shown in Table 3[2]. The analysis and design textbook listed by each respondent was cross checked against the method taught and was found to be consistent. Table 3 shows that the proportion of schools that teach only the structured approach went down from 23% in 2002 to 16% in 2005; whereas those teaching only the OO methodology went up slightly from 4% to 10% among the respondents during the same time period.

[1] 4 of the 31 respondents of the 2005 survey did not offer a programming language course but required their students to take programming classes from the Computer Science department.

[2] While it is possible to obtain additional insight into the pattern of change by tracking the schools that modified their curriculum between the two surveys, we decided not to collect this data to preserve the anonymity of the schools that participated in our survey.

Table 3. Analysis and Design

| | November 2002 (n=47) | | January 2005 (n=31) | |
|---|---|---|---|---|
| Method Covered | Count | Percentage | Count | Percentage |
| Only SAD | 11 | 23 | 5 | 16 |
| Mostly SAD, Intro to OOAD | 27 | 57 | 19 | 61 |
| Mostly OOAD, Intro to SAD | 7 | 15 | 4 | 13 |
| Only OOAD | 2 | 4 | 3 | 10 |

We divided the responses into two groups based on the primary focus of instruction. Responses marking "Only structured analysis and design" or "Mostly structured analysis and design with introduction to OO analysis and design" were categorized as Structured, and the rest were categorized as Object Oriented. The results, shown graphically in Figure 2, suggest that the proportion of curricula with an OO focus has only marginally increased from 19% in 2002 to 23% in 2005. We could not reject the hypothesis that the percentage of schools

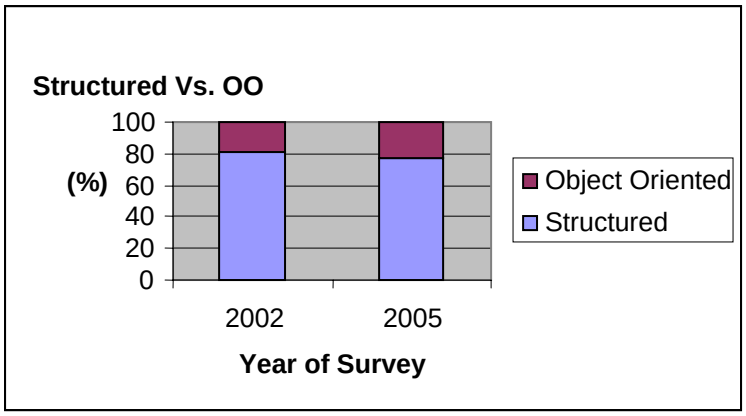


Figure 2. Analysis and Design Methods Taught

teaching OOAD remained the same between 2002 and 2005 (p-value = 0.713 using chi-square test of change in proportions). While the trend in the number of schools offering instruction in OOAD is increasing , it is quite surprising to us that even today more than three quarters of the schools continue to teach the structured approach as the primary focus of their analysis and design courses.

## III. A CASE FOR TEACHING OOAD

We believe that IS departments should consider switching the focus of the analysis and design course from structured analysis to OOAD. In this section, we make a case for teaching OOAD based on the following factors:

- Demand for OO skill in the workforce;
- Mismatch between the IS programming and Analysis & Design curricula;
- Cognitive difficulties in retraining to learn OOAD; and
- Trend in software development methods.

## DEMAND FOR OO SKILL

One of the responsibilities of the IS academic discipline is to ensure that IS graduates possess the right knowledge and skills to meet the industry demand. OO technologies started becoming popular in the 1990s. Some of the benefits attributed to OO development, such as incremental development and facilitation of reuse [Basili et al. 1996], helped it gain wide acceptance in the business software development community. It became the technology of choice of software developers [Schambach and Walstrom 2002]. We believe that schools, therefore, are obligated to ensure that their graduates have adequate OOAD knowledge and skill. Our 2005 survey found that schools updated their curricula to meet the demand for OO programming skill (Section II). However, a wide skill gap still exists in analysis and design.

## MISMATCH WITHIN THE IS CURRICULUM

The transition from structured analysis and design to structured programming in a language like COBOL is relatively seamless because the abstractions they deal with (i.e. functions) and the philosophy of decomposition (primarily top-down) are consistent in both. The fundamental abstraction in OO languages is a class and the nature of decomposition is not based on functionality or procedures, but on the relationships between classes and the distribution of behavior among them. Hence, the translation of structured analysis and design to an implementation in an OO language is unlikely to achieve the advantages of truly object oriented applications, such as flexibility, high cohesion, low coupling, and modularity. OO concepts such as class, encapsulation, inheritance, and composition, which are taught in the OO programming course and are useful in learning OO analysis [Parsons and Wand 1997], are not semantically consistent with structured analysis and design techniques such as Data Flow Diagrams.

In our 2005 survey (Section II), only 11% of the respondents offer a course in COBOL programming (the only procedural language being taught), whereas every school offers at least one course in object oriented programming. However, only a small fraction of these schools (23%) are teaching OOAD. Due to the inherent conceptual mismatch between OO programming and structured analysis and design, students taking this combination of courses are likely to encounter difficulties in relating the knowledge acquired in one to the other. This mismatch within the IS curriculum may be overcome by making OO the primary focus of analysis and design.

### Cognitive difficulty in learning OOAD

One may argue that the skill gap among analysts can be bridged by retraining IS graduates on OOAD. After all, training is an on going effort in industry, and the IS graduate has to continually learn new skills to keep abreast of the latest developments in IT. Unfortunately, this argument does not take into consideration the fact that developers trained in the structured approach find learning the OO approach difficult. While training in an OO language is necessary, it is not sufficient to accommodate the new cognitive skills required for successful OO development [Cockburn, 1998].

OO was initially promoted as a technology that allows the modeler to naturally represent real world objects. This natural mapping between the real world object and its corresponding modeling construct was expected to reduce the cognitive burden of the modeler and simplify the modeling process. Practitioner experience and subsequent research showed that OO technologies are difficult to learn and use [Sheetz et al. 1997]. The difficulties encountered in learning OO and the perceived complexity of OO systems may be attributed to problem decomposition. Decomposition is a key cognitive activity that influences the understanding and conceptualization of the problem, its representation, and the solution that follows [Tegarden and Sheetz, 2001]. Problem decomposition in an OO system typically involves

- identifying classes,
- encapsulation,
- the assignment of responsibilities/behaviors to classes,

- the conception of relationships (e.g. “a kind of” versus “a part of”, whether to use inheritance or composition/delegation), and
- polymorphism.

Abstraction and shared behaviors are salient characteristics of OO [Rosson and Alpert 1990; Sheetz and Tegarden 2001], both of which require a different way of thinking than in the structured approach. Larman [2005, pg. 6] observes that, “A critical ability in OO development is to skillfully assign responsibilities to software objects.” Table 4 summarizes some of the conceptual and cognitive differences between OO and the structured approach.

Structured analysis and design with its emphasis on processes fosters a mindset that hinders, rather than facilitates, the conceptualization of systems in terms of objects and their interactions [Rosson and Alpert 1990]. Sircar et al. [2001] showed a large conceptual gap separating structured analysis and design from OOAD, which may explain the steep learning curve encountered in retraining analysts to learn OOAD. Thus, by training our graduates in the structured approach we are only making it more difficult for them to learn OOAD, which is in higher demand.

Table 4. Key Conceptual and Cognitive Differences between OO and Structured Approaches

| **Characteristics** | **Object Oriented** | **Structured** |
|---|---|---|
| Abstraction(s) | Class, framework, component, patterns | Procedures |
| Decomposition | Classes and their interactions – what emerges is a web of classes | Top-down functional decomposition – what emerges is a tree-like hierarchy of functions |
| Hierarchy of interest | Inheritance structure showing classes and their subclasses | Functions and their sub-functions |
| Domain Focus | Semantic richness – emphasis on *what* concepts, problem-solving strategies, and cognitive aspects | Syntactic aspects – focus on processes, procedures (*how*), inputs, outputs, data flows |
| Representation/Diagrams | Package, Class, Object, Sequence, Communication, Component, Deployment, Statechart, Activity Diagrams | Context, Dataflow, Entity-Relationship diagrams, Structured Chart |

## TREND IN SOFTWARE DEVELOPMENT METHODS:

The software development community is constantly seeking new and better ways to develop software. As a result, software development methods are continually undergoing changes. Structured approaches that dominated in the 1970s and 1980s, gave way to OO approaches in the 1990s. Currently proponents of agile methodologies claim they found a better alternative to the traditional plan-driven approach to software development [Cockburn and Highsmith 2001; Highsmith 2002; Larman 2003]. The standard books on agile methodologies show that a good grasp of OO principles is a prerequisite for building high-quality, flexible systems that are resilient to changes [Beck 1999; Fowler 1999; Martin 2003]. While the effectiveness and usefulness of agile methodologies remains to be proven, the benefit of incremental development is quite evident, especially in the context of large and complex information systems. Class libraries made reuse common practice and patterns extended the notion of reuse beyond code reuse (for example, [Richter 1999]). Shorter cycle times and the demand for higher quality by end-users are moving the software industry in the direction of componentization [Szyperski, 1998; Wallanau, et al. 2002]. This trend in software development towards incremental development, reuse, and componentization favors OO technologies.

Based on the arguments presented in this Section, we believe the need to integrate OOAD into our curriculum is urgent so that our students are ready to meet today's skill demand and to master the future wave of IS development.

## IV. MAKING THE TRANSITION TO OOAD

Although bringing a new technology to the classroom is not a new experience for many IS academics, it requires careful planning. In this section, based on our experience, we provide helpful hints on how to facilitate the transition from structured analysis and design to OOAD.

The most critical resource of an IS program is a well trained faculty [Gorgone et al. 2003]. They emphasize the importance of providing opportunities for professional development of faculty members to keep abreast of technological advances. A key factor in integrating OOAD into the IS curriculum is retraining faculty members teaching structured analysis and design. While this hurdle is not insurmountable, the required retraining effort should not be underestimated. Retraining to learn OOAD can be difficult (section III). Therefore, attending professional training classes and industry internships should be considered to facilitate the learning process.

Even though analysis and design forms part of the core IS curriculum, only a small fraction of IS researchers are actively engaged in investigating issues related to software development [Bajaj et al. 2005]. Bajaj et al [2005] call this phenomenon "the SA&D teaching-research gap." We found no evidence to relate the continued domination by structured methodologies with the SA&D teaching-research gap. It is, however, reasonable to conjecture that professors active in researching software development are more likely to integrate OO methodologies into their courses. While we do not expect every instructor of analysis and design to become an active researcher in this area, narrowing the SA&D teaching-research gap should certainly help many instructors to update their courses with the latest methodologies.

Computing resources and laboratory facilities play a key role in IS instruction [Gorgone et al. 2003]. The Analysis and Design course often requires the use of a CASE tool. Transitioning from structured to OOAD requires updating the CASE tool available in the computing laboratory for student use. Acquiring and installing the software requires careful planning and possible financial commitment. Appendix II provides information on CASE tools for use in an OOAD course.

Instructional material is a key resource for the instructor. Creating new material can consume a substantial amount of an instructor's time when he/she starts preparing for a new course involving a new technology. Analysis and design courses usually make use of cases for student projects and assignments. Preparing such cases is an additional burden on the first time instructor. Finding a good textbook on OOAD with examples appropriate for business students was a difficult task a few years ago. Most early books used examples from computer science and engineering. Several textbooks authored by IS faculty members are now available. These textbooks and the accompanying instructor's resources are helpful to instructors.

The best resource for an instructor planning to adopt OOAD is an experienced colleague. Several such experienced people are in the IS academic community. The Special Interest Group on Systems Analysis and Design (http://teweiwang.net/sigsand), an Association for Information Systems SIG, can take a leading role to facilitate sharing of experience and instructional resources among analysis and design instructors.

## V. CONCLUSION

OO technologies are widely accepted in the IS developer community. Our 2005 survey of AACSB accredited IS programs found that all respondents offer at least one course on OO programming, but only a quarter offer courses on OOAD. This disparity suggests the need to take a closer look at what is taught in analysis and design courses. In this article we argue in

favor of updating the analysis and design course to make OOAD its main focus. Besides the current demand for OO knowledge and skill, future trend in software development methodologies also favor the OO approach. Based on our experience, we provided some helpful hints to analysis and design instructors on how to facilitate the transition to OOAD.

*Editor's Note*: This paper is one in a series of articles in the Research in Information Systems Analysis and Design series, guest edited by Juhani Iivari, and Jeffrey Parsons. Alan Hevner served as the CAIS departmental editor for the series. Some of the papers in this series are being published in JAIS and some in CAIS; the choice depending on the topic and approach of the paper. This paper was received on March 1, 2005. It was with the author for 2 revisions and was published on December 8, 2005.

## REFERENCES

Bajaj, A., Batra, D., Hevner, A., Parsons, J., and Siau, K. 2005. "System Analysis and Design: Should We Be Researching What We Teach?" *Communications of the AIS*, April 2005, 15(27), pp. 478-493.

Basili, V.R., Briand, L.C., and Melo, W.L. 1996. "How Reuse Influences Productivity in Object-Oriented Systems," *Communications of the ACM*, (39) 10, pp. 104-116.

Beck, K. 1999. *Extreme Programming Explained: Embracing Change*. Reading, MA: Addison-Wesley.

Cockburn, A. 1998. *Surviving Object-Oriented Projects: A Manager's Guide*, Reading, MA: Addison Wesley Longman, Inc.

Cockburn, A. and Highsmith, J. 2001. "Agile Software Development: The Business of Innovation," *IEEE Computer*, (34)9, September, pp. 120-127.

Couger, J.D., Davis, G.B., Dologite, D.G., Feinstein, D.L., Gorgone, J.T., Jenkins, A.M., Kasper, G.M., Little, J.C., Longenecker, H.E., Valacich, J.S. 1995. "IS'95: Guideline for Undergraduate IS Curriculum," *MIS Quarterly*, September 1995, (18)3, pp. 341-359.

Fowler, M. 1999. *Refactoing: Improving the Design of Existing Code*. Reading, MA: Addison-Wesley.

Gorgone, J.T., Davis, G.B., Valacich, J.S., Topi, H., Feinstein, D.L., and Longenecker, H.E. 2003. "IS 2002 Model Curriculum and Guideleines for Undergraduate Degree Programs in Information Systems," *Communications of the AIS*, (11)1, pp. 1-53.

Highsmith, J. 2002. *Agile Software Development Ecosystems.* Boston, MA: Addison-Wesley.

Larman, C. 2005. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, (3rd edition) Upper Saddle River, NJ: Prentice Hall PTR.

Larman, C. 2003. *Agile and Iterative Development: A Manager's Guide*, Reading, MA:Addison-Wesley.

Lee, D.M.S., Trauth, E.M., and Farwell, D. 1995. "Critical Skills and Knowledge Requirements of IS Professionals: A Joint Academic/ Industry Investigation," *MIS Quarterly*, (19)3, pp. 313-340.

Marion, W. 1999. "CS1: What Should We Be Teaching?" *SIGCSE Bulletin*, December (31)4, pp. 35-38.

Martin, R. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ: Prentice Hall.

Mitchell, W. 2001. "A Paradigm Shift to OOP Has Occurred … Implementation to Follow," *Journal of Computing in Small Colleges*, May (16)2, pp. 95-106.

Parsons, J. and Wand, Y. 1997. "Using Objects for Systems Analysis," *Communications of the ACM*, 40(12), December, pp. 104-110.

Richter, C. 1999. *Designing Flexible Object-Oriented Systems with UML*, Indianapolis, IN: Macmillan Technical Publishing.

Rosson, M. B. and Alpert, S. R. 1990. "The Cognitive Consequences ofObject-Oriented Design," *Human-Computer Interaction*, (5) 4, pp. 345-379.

Schambach, T.S., and Walstrom, K.A. 2002-2003. "Systems Development Practices: Circa 2001," *Journal of Computer Information Systems*, (43)2, Winter, pp. 87-92.

Sheetz, S.D., Irwin, G., Tegarden, D.P., Nelson, H.J., and Monarchi, D.E. 1997. "Exploring the Difficulties of Learning Object-Oriented Techniques," *Journal of Management Information Systems*, (14)2, Fall, pp. 103-131.

Sheetz, S. D. and Tegarden, D. P. 2001. "Illustrating the Cognitive Consequences of Object-Oriented Systems Development," *The Journal of Systems and Software*, (59)2, pp. 163-179.

Sircar, S., Nerur, S.P., and Mahapatra, R. 2001. "Revolution or Evolution? A Comparison of Object-Oriented and Structured Systems Development Methodos," MIS Quarterly, December (25)4, pp. 457-471.

Szyperski, C. 1998. *Component Software: Beyond Object-Oriented Programming*, (1st edition), Reading, MA: Addison-Wesley.

Tegarden David P. and Sheetz D. Steven. (2001). "Cognitive Activities in OO Development," *International Journal of Human-Computer Studies*, (54) 6, pp. 779-798.

Wallnau, K.C., Hissam, S.A., and Seacord, R.C. 2002. *Building Systems from Commercial Components*. Reading, MA: Addison-Wesley.

## APPENDIX I: THE 2002 SURVEY INSTRUMENT

Dear Professor:

We are seeking information on Programming, and Analysis & Design courses offered to undergraduate Information Systems majors in AACSB accredited programs. We would greatly appreciate your response to the following 3 questions about your curriculum:

1. Which Programming Course(s) is taught in your department? Please list all that are appropriate.

a) COBOL
b) Visual Basic
c) C
d) C++
e) JAVA
f) VB.Net
g) Any other?

2. Which analysis and design method is covered in your Analysis & Design Course? Please select the most appropriate response from the following list:

a) Only Structured analysis and design
b) Mostly Structured analysis and design with introduction to Object Oriented (OO) analysis and design

c) Mostly OO analysis and design with introduction to structured analysis and design
d) Only OO analysis and design

3. Which textbook do you use in your analysis and design course? (Please list the textbook title and name(s) of author(s)).

You may forward the message to the appropriate person in your department in case you do not have the information to respond to these questions.

Thank you for your time.

## APPENDIX II. OOAD PACKAGES AVAILABLE FOR UNIVERSITIES

1. Argo UML (http://argouml.tigris.org/)

Argo UML is one of the first open source development UML packages that became available. It is platform independent. It currently supports the UML version 1.3 Meta-Model. Eight of the nine UML diagrams are supported. It can perform both forward and reverse engineering. It does not support a program development environment.

2. Omondo Eclipse UML (http://www.omondo.com/index.html)

EclipseUML Free Edition and EclipseUML Studio are visual modeling tools, natively integrated with Eclipse 3.1 and JDK 5. Eclipse features include team support, bidirectional code synchronization, native CVS integration, class and sequence diagram reverse engineering from Java byte code, database modeling and deployment. The product requires the installation of the free Eclipse IDE (http://www.eclipse.org/). The analysis and design interface is tightly integrated with the program development environment. JUnit testing is tightly integrated with Eclipse. Eclipse UML requires the user to increase the amount of JVM memory used by Eclipse.

3. Visual Paradigm Community Edition by Visual Paradigm (http://www.visual-paradigm.com/productinfovpumlce.php)

Visual Paradigm supports all UML 2.0 diagrams. Other features include real time model validation, resource concentric interface, the ability to copy diagram elements to the Windows clipboard, use case textual analysis and textual analysis for sequence diagrams. The modeling environment is not closely integrated with the program development environment.

4. BOUML (http://bouml.free.fr/)

BOUML is a sophisticated free UML toolset written by Bruno Pages. The source code is freely available, and can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation. This package supports most of the necessary UML tools. However, the user interface is not intuitive and takes some time to learn.

## ABOUT THE AUTHORS

**RadhaKanta Mahapatra** is Associate Professor of Information Systems at the University of Texas at Arlington. He holds a a Ph.D. in Information Systems from Texas A&M University. His research interests include software development methodologies, knowledge management, data mining, web-based end-user training, and IT management. His research publications appear in such journals as *MIS Quarterly*, *Communications of the ACM*, *Decision Support Systems*, *Information & Management*, and *Journal of Database Management*. He received the *Distinguished Research Publication Award* and the *Distinguished Professional Publication Award* from the College of Business Administration of the University of Texas at Arlington.

**Sridhar Nerur** is Assistant Professor of Information Systems at the University of Texas at Arlington. He received his PhD from the University of Texas at Arlington. His research interests

include software development, citation analysis, reuse, maintenance, and philosophical aspects of systems development. His publications appear in the *MIS Quarterly*, *Communications of the ACM*, *Information Management & Computer Security*, and in various conference proceedings.

**Craig Slinkman** received his PhD from the University of Minnesota in quantitative methods and information systems in 1984. Dr. Slinkman's teaching interest is in the areas of object-oriented technology, development, and database management. His current research interests are in object oriented development. His publications appear in *MIS Quarterly, Journal of Nursing Research, Journal of High Technology Management, Communications in Statistics, and Texas Journal of Rural Health.*

Communications of the Association for Information Systems

ISSN: 1529-3181