# Skin vs. Guts: Introducing Design Patterns in the MIS Curriculum

*Elizabeth Towell and John Towell*
*Carroll College, Waukesha, WI, USA*

etowell@cc.edu jtowell@cc.edu

## Executive Summary

Significant evidence suggests that there is increasing demand for information technology oriented students who have exposure to Object-Oriented (OO) methodologies and who have knowledge of the Unified Modeling Language (UML). Object-oriented development has moved into the mainstream as the value of reusable program modules is becoming more evident. Building blocks such as software components and application frameworks allow organizations to bring products to market faster and to adapt to changes more quickly.

When only one analysis and design course is required, the study of object-oriented analysis and design is often minimal or left out of the Management Information Systems curriculum. This paper argues for an increased emphasis on object-oriented techniques in the introductory Systems Analysis and Design course. Specifically, it provides guidance for faculty who are considering adding object-orientation to a systems analysis and design course where existing content leaves little time for new topics.

In particular, the authors recommend the introduction of two successful, generalized design patterns in the context of adaptation of a software development framework. Where time is limited, coverage of such recurring software problems cannot be comprehensive so it is proposed that students explore two design patterns in depth as opposed to a cursory review of more patterns. Students must first learn or review the OO vocabulary including terms such as classes, inheritance, and aggregation so that the appropriate term can be used to discuss the UML relationships among the object model components. Students should also draw analogies to patterns in other disciplines. This is important so that students understand that this exploration is drawn from the more general wisdom of reusable knowledge and is not tied to a passing technological fad. Finally, students should be presented with a case study based on the adaptation of an application development framework based on the use of the Decorator and Strategy design patterns.

Tentative conclusion are drawn by authors based on anecdotal evidence indicating that students believe that this coverage gives them a better understanding of OO concepts in general and design patterns specifically. Students that have a background in Java suggest that the exercise gave them a better understanding of the use of interfaces (classes that have unimplemented methods). The authors intend to capture this data more formally in the future and hope to develop a valid pre- and post- test instrument to determine the extent to which students can apply what they have learned.

**Keywords**: Decorator Pattern, Design Patterns, MIS Curriculum, Object-Orientation, Software Frameworks, Strategy Pattern, System Analysis and Design



**Editor: Linda Knight**

# Introduction

System development is a crucial function in the modern organization. Likewise, an introductory course in Systems Analysis and Design remains a critical component in a Management Information System (MIS) degree program. (Most MIS programs also include these components: Fundamental Concepts of Information and Computer Technology, Computer Hardware and Software, Project Management, Database Design, Telecommunications and Networking, Introductory Programming, and Information Systems Management.) Broadly speaking, the Analysis and Design course involves techniques from four skill categories: interpersonal skills, technical skills, analytical skills, and communication skills (Misic and Russo, 1999). Detailed content analysis, for twenty-nine of the leading textbooks in this area, is provided by Misic and Russo (2000).

An old saying suggests that "in theory, there's no difference between theory and practice, but in practice there is…" Like Woratschek (1999), academicians who spend a little time in the trenches often learn that formal system analysis and design techniques are sparsely used in many organizations. This gap between tasks, tools, activities and approaches that are taught in the MIS curriculum and those that are used, or viewed as useful, by practitioners has been documented (Misic and Russo, 1999). There are many valid reasons why gaps like these exist. Educators know that activities used to *learn* a process are often different from those required to *do* the process. They realize that instruction should involve formative (best practices) as opposed to normative (usual practices). Educators also often feel responsible for leading the change in preparing future systems analysts.

Nowhere, however, does the gap between education and industry seem more profound and inexplicable than in the teaching of object-oriented (OO) principles. The OO approach offers clear advantages of reusability, extendibility, and portability. On the one hand, we see the Internet driving the future and with it the demand for OO information technology skills such as proficiency in the OO programming language Java (Brandel, 2000; Enticknap, 2001; Watson, 2000). In some firms, staff increases up to 40% are expected for people with knowledge of OO methodologies and UML (Goff, 1998). Object-oriented development has moved into the mainstream with the widespread adoption of the Common Object Request Broker Architecture (CORBA), Microsoft's .NET Initiative, and Sun's Enterprise Java (J2EE) (Langley, 2001; Sullivan, 2001). On the other hand, OO analysis and design seems to play a minor role in the current MIS curriculum. Misic and Russo (2000) indicate that of 29 systems analysis and design textbooks studied, 26 (or 90%) of them devoted less than 5% of the text to OO analysis and design techniques. McLeod (1996) indicates that "instructors tend to favor the traditional process-oriented methodologies over the more recently developed ones that emphasize data and objects…decisions strongly influenced by the instructor's experience."

This paper focuses on an object-oriented technique commonly used in software design, namely the identification and application of design patterns. We discuss the introduction of design patterns in the System Analysis and Design course, providing a concrete example involving two patterns, the Decorator pattern (the skin) and the Strategy pattern (the guts).

# Design Patterns and Software Frameworks

Few complex software systems today are built from scratch using the so-called Greenfield engineering strategy (Bruegge and Dutoit, 2000). Many more projects are evolutionary, involving the redesign and re-implementation of an existing system and/or its interface, triggered by new technology or new information. Reusable program modules are beginning to be recognized as having tremendous strategic value as managers face re-engineering failures at a rate as high as 70% (Fayed, 2000). "By 2003," says Michael Blechar of the Gartner Group, "at least 70% of the total number of new applications will be built primarily from 'building blocks' such as software components and application frameworks, increasing both products' speed to market and enterprises' ability to cope with change." He continues, "IS

organizations must have a strategy for the inevitable shift to component-based development. Components in the form of design patterns, application frameworks and business and technical components can significantly increase quality and productivity in development through reuse."

A *design pattern* is a successful, generalized solution to a recurring software problem. Scores of popular design patterns have been catalogued and are now in regular use (Gamma et al., 1995). A *framework* is a reusable, somewhat complete, application that can be specialized to produce a custom application. Application frameworks are targeted to a particular technology or specific application domain. Design patterns and frameworks take advantage of the distinguishing characteristics of an OO environment: abstraction, polymorphism and inheritance. Both facilitate code reuse by capturing successful software development strategies. Design patterns and frameworks together with class libraries and components are in widespread use in all business sectors to increase software quality and to reduce development effort.

## Introducing Students to Design Patterns

Design patterns can be classified into three basic categories: creational patterns, structural patterns, and behavioral patterns. Creational patterns help make a system independent of how objects are created and represented. An example of this pattern, the Abstract Factory, can be used to shield an application from a specific windowing style or operating system, allowing, for instance, buttons and scroll bars to run uniformly on a range of platforms. Structural patterns have to do with how program components are integrated to form larger systems. An example of such a structural pattern is the Adaptor pattern, which makes one interface conform to another. Behavioral patterns are concerned with the assignment of responsibility and the communication between program components. The Observer pattern enables the maintenance of consistency across the states of one publisher and many subscribers. All design patterns have the following essential elements: a name, a problem, a solution, results and trade-offs. When published, design patterns also tend to include known uses, example code, and related patterns (Gamma et al., 1995).

Designing software is difficult and designing reusable software components is even more difficult. Expert software engineers, like designers in many fields recognize that you should not solve every problem from first principles. Experienced designers recognize similar problems and modify and reuse solutions that have worked for them in the past. Design patterns can help analysts make designs that are more flexible, reusable and understandable, but understanding design patterns requires patience and the proper foundation.

The next section provides an overview of two design solutions and how they might be used to introduce design patterns in the design portion of a System Analysis and Design class. The first of the two solutions involves a structural pattern called the Decorator pattern (changing the "skin" of an object). The second is related to a behavioral pattern called the Strategy pattern (changing the "guts").

## Patterns in Two Weeks

Which content should be given up if OO concepts are to be introduced? Where does an introduction to OO concepts best fit? How can OO be reconciled with a course that is based on the traditional system development life cycle and the waterfall approach to analysis and design? These are difficult questions but important ones.

Many textbooks and, it is supposed, many instructors cover topics in the Introductory System Analysis and Design course that are also covered in other courses (for example: database design, project management, organizational style, sampling techniques, questionnaire design, decision-making behavior). This is the best place to look for course components that can be reduced or eliminated. An introduction

to object orientation is simply too important to be left out, especially if there is only one course in analysis and design.

Different projects require different processes and dependencies. Students exposed to the waterfall method should realize that there are other life cycle models and the best ordering of developmental activities depends a great deal on the nature of the project. The introduction of object-oriented projects offers a good opportunity to discuss other life cycle models, including the Unified Software Development Process which is more iterative in nature.

## *Week 1: Review of Object-Oriented Concepts*

Initially students should be provided with a review of basic OO concepts. (Our students gain OO experience through a sequence of activities in an object-oriented environment known as a MOO (Towell, 2000). Some of them, but not all have completed a course in an object-oriented programming language.) There are several introductory lectures in the public domain. A particularly good one can currently be found at http://ei.cs.vt.edu/~kafura/java/ (Chapter 1: Basic Concepts). This tutorial by Dr. Dennis Kafura provides text, graphics, and simple exercises on abstraction, separation, classes, objects, abstract interfaces, composition and generalization. He defines "pattern" in the following way:

*"A pattern expresses a general solution (the key components and relationships) to a commonly occurring design problem. The attributes and behavior of the individual components are only partially defined to allow the pattern to be interpreted and applied to a wide range of situations. For example, a "wheeled vehicle" pattern might be defined in terms of the components "wheel," "axle," "frame," "body" and "power source." The pattern would also show how these components would be arranged in relation to each other (e.g., the axle must connect two wheels). The pattern could be interpreted in many different ways to solve particular problems that differ in their requirements for speed, durability, payload, fuel source, available materials, and other factors. Examples of the wheeled vehicle pattern are "automobile," "horse-drawn carriage," "ox cart," "moon buggy" and many others."*

## *Week 2: Exploring Patterns Further*

### Writing exercise

Students should be asked to write several specifications for patterns from various disciplines. There are countless occupations to draw from including civil engineers, chefs, air traffic controllers, gardeners, process control engineers, packaging experts, fashion designers, and architects. Like the "wheeled vehicle," each pattern should describe a general problem, which occurs frequently in a given environment and a core solution that is refined to solve a particular case. Each pattern should have the following format:

*Context:*
*Pattern Name:*
*Problem:*
*General Solution:*
*Examples of Specific Solutions:*

Instructors should provide at least one example pattern, for example:

*Context:* Building construction
*Pattern Name*: Insulators
*Problem*: Loss of energy

*General Solution*: Materials that are poor conductors can prevent loss of energy by surrounding energy producers and conductors

*Examples of Specific Solutions:* Fiberglass insulation in walls reducing loss of warm air and polyurethane insulation around pipes reducing loss of hot water heat.

## Class discussion

Encourage students to find ways to categorize patterns they have written. One obvious taxonomy has to do with context. Attempt to categorize patterns as creational (making a system independent of how its objects are created), structural (how objects can be put together to form a larger structure), or behavioral (how objects act, cooperate and communicate). For instance, wheeled vehicles and insulators are both structural patterns, that is, having to do with putting objects together to realize new functionality. Ask students to look for examples of libraries of patterns. This might include books on first aid, home repair, or drawing animals. Finally, have students read the following case study involving programming patterns and then ask them to describe other possible applications for the patterns discussed.

## Case study

ABC Company sells a variety of full-size and near full-size wrestling dummies to schools, sport teams, clubs, sporting good stores, and individuals. They have been in business approximately five years with annual revenues now exceeding $10M.

ABC Company has decided to develop a set of web-based applications for their growing customer base. After considerable investigation, ABC decides that they will use an application development framework comprised of a library of server-side Java components. These components will provide the necessary database access, secure and high-performance session management, background job handling, and event notification and logging. Using this framework, ABC developers will be able to focus more on the necessary business logic and layout of the user interface.

After exploring the framework demonstrations from the vendor, ABC determines that they want to modify two of the supplied components. ABC would like to add a logo border to the user login component. Future updates to the user login component by the vendor, however, should not cause problems with the border. In turn, the border should be transparent to other related framework components that depend on user login, including components related to user registration, file downloads, and user groups. The border should be easily modified. Another desired modification involves the syslog component. ABC would like to occasionally vary the algorithm that is used to compute user connect time. They have evaluated several computational methods involving variables such as idle time, CPU speed and usage, clock time, caching level, pages accessed and keystrokes. They would like to define a family of algorithms and make them interchangeable. ABC Company is able to accommodate these changes through the use of two design patterns.

## The Decorator Pattern

The decorator pattern is a structural pattern. It allows for the enhancement of the user login component in such a way that it is transparent to other objects that interact with user login. User login is enclosed in a "decorator" object that has the same interface as user login. The decorator component forwards requests to user login and performs additional actions before forwarding. We can think of the decorator as a skin over the user login object (Figure 1).

The class model for the Decorator pattern is shown in Figure 2. The class LOGIN SCREEN is an abstract class that defines the public protocol (allowable methods) for a login screen. The CONCRETE LOGIN SCREEN is a subclass of LOGIN SCREEN that has fully implemented methods, and which can be instantiated. LOGO DECORATOR is also a subclass of LOGIN SCREEN. Therefore it has the

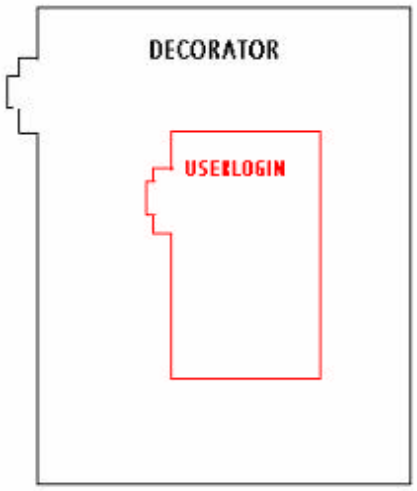


**Figure 1** User Login Enclosed in a Decorator Object

same interface as the CONCRETE LOGIN SCREEN. An object of type LOGO DECORATOR is instantiated by passing two arguments to a constructor method, an object of type LOGIN SCREEN and a real number representing the border width. The CONCRETE LOGIN SCREEN (supplied within the framework) and the LOGO DECORATOR (created and maintained by ABC) can be changed independently. LOGO DECORATOR adds capability to any object of type LOGIN SCREEN. The LOGO DECORATOR is composed of an object of type LOGIN SCREEN and additional features. Note that this means the decoration could be added twice.

A non-software example of a decorator would be lights and ornaments on a Christmas tree. The ornaments do not change the tree, nor does the tree need to be aware of its ornaments. As an example of added functionality, with the addition of lights, you are then able to "light up" the tree (Duell, 1997).

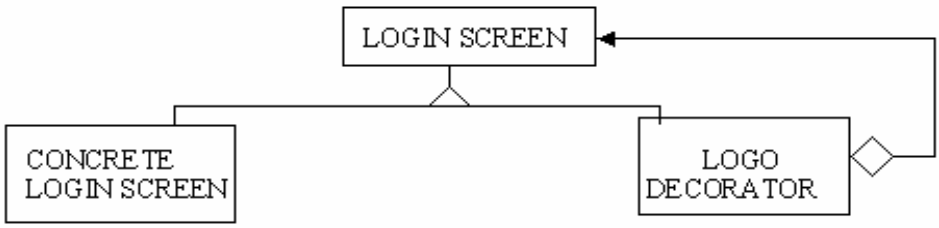


**Figure 2** The Object Model for the Decorator Pattern

## The Strategy Pattern

The strategy pattern is a behavioral pattern. It provides a standard for communication among objects. It allows the ABC Company to use different algorithms at different times to compute connect time. The strategy pattern is actually an algorithm interface, defined for the connect time operation. All of the concrete implementations for calculating connect time are designed to adhere to this interface. Syslog can then depend upon the interface (what is needed and what is returned by this computation) without concern for the details of how it will be carried out. We imagine "strategy" as a pattern for changing the guts of an object (see Figure 3).

The class model for the Strategy pattern is shown in Figure 4. CONNECT TIME STRATEGY is an abstract class that defines the interface for the algorithm, which computes connect time (what must be provided and what will be produced). Each of the CONNECT TIME CONCRETE STRATEGYs provides a different implementation for calculating connect time. Each of the CONCRETE STRATEGYs adheres to the abstract interface, although they may not all actually use all the parameters that are supplied. The SYSLOG class is composed of a CONNECT TIME STRATEGY as well as other things.

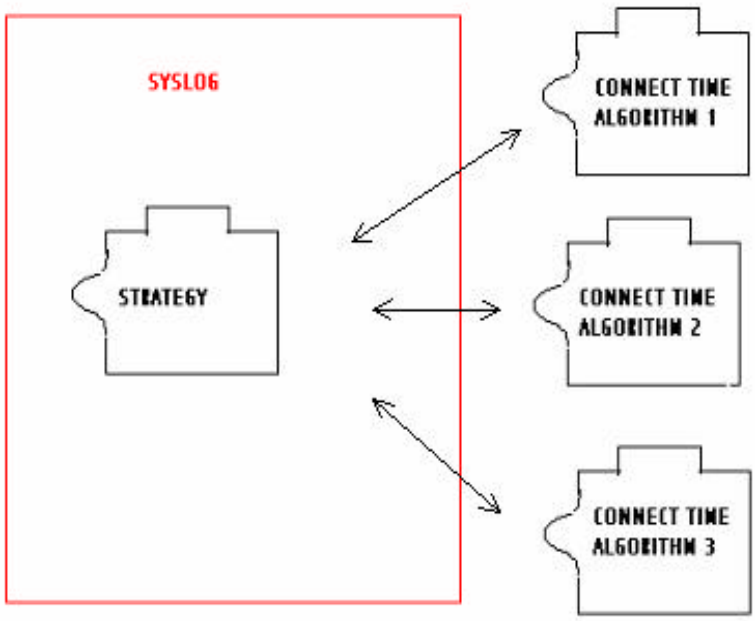


**Figure 3** The Strategy Pattern Defines An Interface for the Connect Time Calculation

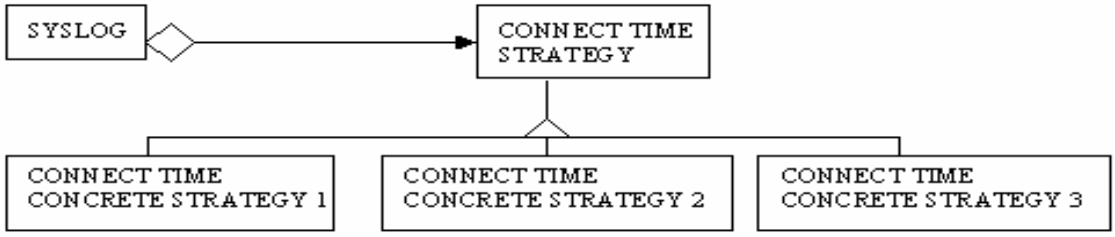


**Figure 4** The Class Model for the Strategy Pattern

A non-software example of the strategy pattern is the contractual arrangement you might have with a landscaper to mow your lawn. You specify the interface: what you will provide (in terms of regular payment) and what you expect (short grass). How the grass actually is mowed can vary over time (the route, the type of machine, the number of people involved, for example).

## Discussion

In a small amount of time within a traditional systems analysis and design course, students cannot become experts in object-oriented design. However, by exploring two design patterns used together with a software development framework, students can begin to appreciate the advantages of object-oriented designs. Two important concepts are demonstrated through the use of this case study and the exploration of these design patterns: 1) Object-oriented *designs* often end up with classes that have no counter-

part in the real world; and 2) Partially implemented abstract classes make a design more flexible and reusable.

Analysts spend more time thinking about a system's application domain and less time thinking about the solution domain. An application domain model captures the user's view of the system. The system model (which is often left to those that implement the system) does a better job of reflecting entities that are part of the system. Students should be made to realize that there will be components in most solutions that have no meaning for the user of the system. By understanding a little about the role of such system components, students may be better prepared to make choices when developing the application domain model. The decorator and the strategy pattern are solution components that will be used to give the ABC Company more flexibility in making use of the application development framework. These objects, themselves, will have no meaning to the users. They are not stand-alone programs, that is, they serve as intermediate program components that are used as an interface to other related objects that are fully implemented. These objects facilitate future changes to the system.

Authors first shared this case study with students in the spring 2002. The students indicated, anecdotally, that they felt they had a better understanding of object-oriented concepts, in general, and design patterns specifically. Students that had a background in Java indicated that they came away with a better understanding of interfaces (a class that has only methods with no instance variables). Students with no object-oriented background claimed that the idea of design patterns (explored in many contexts) gave them a better understanding of the value of reusable code.

Several recommendations are made regarding the introduction of design patterns in the MIS curriculum. First, students should be reasonably familiar with OO concepts such as classes, inheritance, aggregation, abstract data types, and polymorphism before design patterns are introduced. This is important when learning about design patterns because these terms are used to explain the relationships among the object model components. Secondly, students will become more comfortable with design patterns after drawing analogies to patterns in other disciplines. The authors believe this is important because students are then able to realize that the exercise is drawn from more general wisdom about reusable knowledge, not from a passing technological fad. Finally, the authors believe that, when time is limited, students should learn about one or two design patterns in detail as opposed to conducting a cursory study of many. This is based on the experience of the authors who believe that teaching students about technology should focus where possible on a few technological themes as opposed to many details and facts.

Ideally, an entire course should be devoted to OO Analysis and Design. If this is not possible in the curriculum, OO principles should probably be introduced in the traditional course. The case study described here is offered as a single example of the use of object orientation and design patterns to achieve more flexibility when using an existing software development framework. This case study has only been used in one class and the evidence of learning that authors have to date is purely anecdotal. In future offerings, we intend to capture this data formally, assessing the extent to which students believe desired learning outcomes are achieved. We also intend to develop a valid pre- and post- test instrument to determine the extent to which students can apply what they have learned.

## Conclusion

This paper argues that grounding in object-oriented concepts should now be considered an essential component of the MIS curriculum. If a standalone course is unfeasible, instructors should introduce object-oriented concepts and reusable code in the System Analysis and Design course, reducing, if necessary, what is covered elsewhere in their curriculum. As evidenced in the leading textbooks, the initial Systems Analysis and Design course provides broad exposure to a large number of topics. Different authors (and, it is supposed, different instructors as well) cover these topics in varying depth. Many of these topics are covered in other courses as well, and it is therefore less important to cover them in the

analysis and design course (for example: database design, project management, organizational style, sampling techniques, questionnaire design, and decision-making behavior). By removing or minimizing coverage of topics addressed elsewhere in the curriculum, faculty are provided time to introduce object-oriented concepts. This paper provides a concrete example of how this can be accomplished.

This paper focuses on an object-oriented technique commonly used in software design, namely the identification and application of design patterns. We discuss the introduction of design patterns in the System Analysis and Design course, providing a concrete example involving two patterns, the Decorator pattern (the skin) and the Strategy pattern (the guts). These design patterns are used by a fictitious company to modify an off-the-shelf application development framework comprised of a library of server-side Java components. The modifications result in a more flexible set of web-based applications for the company.

The approach taken here is based on the concept that when time is limited, it is more effective to give students a more thorough understanding of two design patterns, rather than attempt to cover a wide variety of patterns. Through the use of the case study detailed here, students develop an overall familiarity with object oriented concepts, gain specific knowledge of two patterns, and come to appreciate the usefulness of partially implemented abstract classes in achieving flexible, reusable designs.

# References


Brandel, M. (2000). "The Top Skills to Watch," *Computerworld*, May 22, p.91.

Bruegge, B. and Dutoit, A. H. (2000). *Object-Oriented Software Engineering, Conquering Complex and Changing Systems*, Prentice Hall, Upper Saddle River, NJ.

Duell, M. (1997) "Non-Software Examples of Software Design patterns," *Object Magazine*, July.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley.

Goff, L. (1998). "Objects of Affection," *Computerworld*, March 23, p. 85.

Enticknap, N. (2001). "Internet Drives Demand for IT Skills," *Computer Weekly*, February 22, p. 24.

Fayad, M. E. (2000). "Introduction to the Computing Survey's Electronic Symposium on Object-Oriented Application Frameworks," ACM Computing Surveys, Vol. 32, Issue 1, p.1.

Kafura, D. (2002). "Object-Oriented Software Design and Construction With Java, Chapter 1: Basic Concepts," Available http://ei.cs.vt.edu/~kafura/java/ downloaded November 19, 2002.

Langley, N. (2001). "Building Blocks for Coders," *Computer Weekly*, April 26, p. 64.

McLeod, R. (1996). "Comparing Undergraduate Courses in Systems Analysis and Design," *Communications of the ACM*, Vol. 39, No. 5, pp. 113-121.

Misic, M. M. and Russo, N. L. (1999). "An Assessment of Systems Analysis and Design Courses," The Journal of Systems and Software, Vol. 45, pp. 197-202.

Misic, M. M. and Russo, N. L. (2000). "Reading Between the Lines: An Examination of Systems Analysis and Design Texts," *The Journal of Systems and Software*, Vol. 50, pp. 65-73.

Sullivan, T. (2001). "Swappable Java-Based Servers Gain Ground," *InfoWorld*, Vol. 23, No. 5, p. 39

Towell, J. (2000), " MOO: An active-learning environment for teaching object-oriented concepts in business information systems curricula," *Journal of Information Systems Education*, Vol. 11, pp. 147-150.

Watson, S. (2000). "The Best Jobs; E-commerce continues to dominate new IT hiring; meanwhile, demand for contractors is dropping fast," *Computerworld*, April 3, p. 46.

Woratschek, C. R. (1999). "Observations and Experiences from the Trenches: A Sabbatical Year in the Business Environment" *Proceedings, Information Systems Education Conference* (ISECON), Chicago, IL October 14-17.

# Biographies

**Elizabeth Towell** is an Associate Professor of Computer Science and Business at Carroll College in Waukesha, WI. She previously was an Associate Professor in Operations Management and Information Systems at Northern Illinois University. She teaches classes in Systems Analysis and Design and Database Design. Her publications have appeared in *Information, Technology and People*, *Mid-American Journal of Business*, *New Horizons in Adult Education*, *Global Information Technology Management*, *Internet Research*, *Presence (MIT)*, *Journal of Computer Information Systems*, the *Computer Education Quarterly*, and the *Journal of Database Management*. She received her PhD from the University of WI at Milwaukee.

**John Towell** is an Assistant Professor of Computer Science at Carroll College in Waukesha, WI. His research interests are in the use of object-oriented virtual environments as teaching/research tools. His areas of teaching expertise are Java Programming, Object-Oriented Analysis and Design, and Advanced Web Design (J2EE). In addition to his teaching and research responsibilities, he is also system administrator for a Linux system on the Carroll College campus that supports an Oracle DBMS, an Apache-Tomccat-Jboss Web Server, and a virtual environment called Carroll MOO. He has a Bachelor of Arts Degree from the University of Colorado and a Ph.D. from Colorado State University.