ISSN: 1545-679X

# Information Systems Education Journal

Volume 4, Number 81 http://isedj.org/4/81/ September 22, 2006

**In this issue:**

## Metaphors, Polymorphism, Domain Analysis, and Reuse: Teaching Modeling in the Object-Oriented Paradigm

**Leslie J. Waguespack, Jr**
Bentley College
Waltham, MA 01254 USA

**Abstract:** Object-oriented programming has become a mainstay of computing curricula over the last decade. Although its industrial promise for improving productivity, particularly by way of enabling extensive reuse, has propelled it to an essential status, it is usually taught in a vacuum of little or no effective modeling theory or practice. In this paper we argue that this vacuum robs most students of their potential to both understand or professionally profit from the complex mass of syntax and class library detail in which they are drowned in most OO development courses. The paper reviews OO-based reuse, the current state of modeling in IS2002 national curriculum and contemporary systems analysis texts, the underlying behavior and metaphor-driven principles of domain modeling and a framework for recovering the reuse benefits of the OO paradigm in IS education.

**Keywords:** modeling, object-oriented modeling, behavior-driven modeling, metaphor-driven modeling, domain modeling, systems analysis and design curricula, IS curricula

---

**Recommended Citation:** Waguespack (2006). Metaphors, Polymorphism, Domain Analysis, and Reuse: Teaching Modeling in the Object-Oriented Paradigm. *Information Systems Education Journal, 4* (81). http://isedj.org/4/81/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2005:* §2332. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/4/81/**

The **Information Systems Education Journal** (ISEDJ) is a peer-reviewed academic journal published by the Education Special Interest Group (EDSIG) of the Association of Information Technology Professionals (AITP, Chicago, Illinois). • ISSN: 1545-679X. • First issue: 8 Sep 2003. • Title: Information Systems Education Journal. Variants: IS Education Journal; ISEDJ. • Physical format: online. • Publishing frequency: irregular; as each article is approved, it is published immediately and constitutes a complete separate issue of the current volume. • Single issue price: free. • Subscription address: subscribe@isedj.org. • Subscription price: free. • Electronic access: http://isedj.org/ • Contact person: Don Colton (editor@isedj.org)



EDSIG activities include the publication of ISEDJ, the organization and execution of the annual ISECON conference held each fall, the publication of the Journal of Information Systems Education (JISE), and the designation and honoring of an IS Educator of the Year. • The Foundation for Information Technology Education has been the key sponsor of ISECON over the years. • The Association for Information Technology Professionals (AITP) provides the corporate umbrella under which EDSIG operates.

# Metaphors, Polymorphism, Domain Analysis, and Reuse: Teaching Modeling in the Object-Oriented Paradigm

Leslie J. Waguespack, Jr., Ph.D.
LWaguespack@Bentley.edu
Computer Information Systems Department, Bentley College
Waltham, Massachusetts, 01254 USA

**Abstract**

Object-oriented programming has become a mainstay of computing curricula over the last decade. Although its industrial promise for improving productivity, particularly by way of enabling extensive reuse, has propelled it to an essential status, it is usually taught in a vacuum of little or no effective modeling theory or practice. In this paper we argue that this vacuum robs most students of their potential to both understand or professionally profit from the complex mass of syntax and class library detail in which they are drowned in most OO development courses. The paper reviews OO-based reuse, the current state of modeling in IS2002 national curriculum and contemporary systems analysis texts, the underlying behavior and metaphor-driven principles of domain modeling and a framework for recovering the reuse benefits of the OO paradigm in IS education.

**Keywords:** object-oriented modeling, behavior-driven modeling, metaphor-driven modeling, domain modeling, systems analysis and design curricula, IS curricula.

## 1. INTRODUCTION

The use of Java, C++, C# and other object oriented programming languages is now commonplace in programming courses in IS curricula. Although one of the greatest motivations for the move from procedural languages to object oriented (OO) programming has been to increase code reuse, a comprehensive philosophy of reuse throughout the software life cycle (requirement, analysis, design and testing models) is largely absent. The study of domain analysis is virtually non-existent in computing curricula although it is at the core of reuse across the full life cycle. Domain analysis is analysis focused on the environment surrounding any particular system with the intent of identifying those business rules and requirements that are common to any applications within the domain. Once achieved, domain analysis enables significant improvements in the economic, reliability and time-to-market aspects of projects that develop additional systems within the same domain – a product line of applications in that domain.

Classically trained IS professionals are not prepared to conduct domain analysis and design for reuse. They have been carefully conditioned to focus on applications rather than upon the domain of any specific application. They are most likely to see software resources as collections of programs as opposed to an integrated model of an organization's information processing, much less as an instance of systems in a domain (Jacobson, Griss & Jonsson 1997). Brooks (Brooks 1987) argues that while there may be no "silver bullet," we can nurture gifted designers. Enhancing requirements analysis, systems analysis and design through reuse and domain awareness facilitates what Brooks calls "elegant solutions."

This paper briefly surveys the current state of OO modeling in the IS2002 curriculum and contemporary systems analysis and de-

sign texts. It summarizes the OO paradigm's key role in achieving industrial-scale, enterprise-wide reuse. It examines the need for OO modeling with a focus on requirement scope to exploit reuse opportunities. The paper describes the behavior-driven abstraction approach, modeling based upon metaphors, that is integral to effective domain analysis; and essential to IS education.

## 2. IS2002 AND MODELING

There are no specific learning units targeting object modeling in IS2002 (Gorgone, Davis, Feinstein, Longenecker 2002). The only reference to object modeling in all of IS2002 is a reference to a 1994 paper on domain modeling (Waguespack 1994) that is not included in the content of the learning units or the course descriptions. The object paradigm per se is not addressed, as the assumption appears to be that programming in object-oriented languages somehow results in object oriented analysis and design practice.

In many IS2002 implementations only the IS2002.7 course addresses modeling specifically and that is usually as one of the forms of specification dialect (ex. DFD, ERD, EER, [more recently] UML). Object oriented analysis is not specified in IS2002 although references to object-orientation imply that it is a departure from "structured" and "event-driven" "design" approaches. There are five IS2002 courses where object oriented modeling would seem to be an essential concept:

2002.5 Programming, Data, File and Object Structures,

2002.7 Analysis and Logical Design,

2002.8 Physical Design and Implementation with DBMS,

2002.9 Physical Design and Implementation in Emerging Environments, and

2002.10 Project Management and Practice

Only in IS2002.5 are OO concepts like inheritance or polymorphism addressed in the learning units. And the focus is on the construction of program mechanisms rather than representing functional requirements.

### Systems Analysis Texts

Not surprisingly the lack of emphasis on domain analysis and behavior-driven modeling in the model curricula is mirrored in texts targeted for the IS2002.7 course. These books are tasked with covering a vast expanse of information system issues including: defining systems, organizational behavior in development, project management, cost/benefit analysis, and (along the way) functional and operational requirements acquisition and specification, "modeling." Table 1 below presents a cursory survey of content coverage in eight currently used texts.

Table 1 – Systems Analysis Text Coverage

| Textbook | Arlow'02 | Harris'03 | Hoffer'02 | Hoffer'05 | Schach'04 | Stumpf'05 | Dennis'05 | Satzinger'02 |
|---|---|---|---|---|---|---|---|---|
| SDLC | N | Y | Y | Y | N | L | L | Y |
| PM | LU | Y | Y | Y | LU | LU | LU | Y |
| DFD | N | L | Y | Y | N | Y | Y | L |
| ER/EER | N | Y | Y | Y | N | N | N | Y |
| UML Use Case | Y | L | L | L | Y | Y | Y | Y |
| UML Class | Y | L | L | L | Y | Y | Y | Y |
| UML Sequence | Y | L | L | L | Y | Y | Y | Y |
| Other UML | Y | N | L | L | Y | Y | Y | L |
| Behavior Driven Modeling | N | N | N | N | N | N | N | N |

Legend
- SDLC – software development life cycle
- PM – project management
- DFD – data flow diagramming
- ER/EER – entity relationship / extended ER

Level of Coverage
- Y – covered (a stand alone treatment)
- N – not covered
- L – limited coverage (introductory only)
- U – covers Unified Process specifically

(*This survey is not intended to criticize the authors, but rather to demonstrate the dearth of treatment that modeling receives in the typical computing student's education.*)

Arlow (Arlow & Neustadt 2002) focuses almost exclusively on UML syntax with limited diversion into the conceptual nature of sys-

tems or SDLC – the minimum necessary to address the *unified process*.

Harris's (Harris 2003) treatment of OO is effectively cosmetic (primarily notation with little paradigm discussion). It is treated almost as a pure syntactic alternative to ER and contains no pedagogical support for behavior-driven modeling.

Hoffer (Hoffer & Valacich 2002) focuses on a traditional SDLC discussion of development with heavy investment in process (DFD) and data (ER) modeling as the driving motivation. The final chapter is a genuflection to object orientation with a not-so-carefully chosen vocabulary for definition and explanation of object-oriented concepts.

Hoffer (Hoffer & Valacich 2005) basically takes from the 3rd edition and distributes the OO content that was the final chapter into the sections where they believe the content is "interchangeable." There is little or no apparent expansion of the treatment of OO either in the modeling or project architecture sense in this update of the previous edition.

Schach (Schach 2004) dives past most discussion of SDLC into the use of the *unified process* and the diagrams therein. It also uses several symbols from (Jacobsen, Griss & Jonsson 1997) (e.g. interface, entity, control objects) reminiscent of component / deployment documentation in the Jacobsen's reuse text. However, the symbols are not supported by significant concept development.

Stumpf (Stumpf & Teague 2005) introduces a domain model as a concept (not common in the others), however the definition / description of it is virtually indistinguishable from an ER model of the application space. Modeling appears fixated on the externally visible actions of the current system implementation rather than any focus on business rules.

Dennis (Dennis, Wixom & Tegarden 2005) offers the most complete content among these texts regarding object-oriented analysis. The treatment appears to focus on reproducing perceived system interface function rather than discovering underlying business rules to guide the domain modeler.

Satzinger (Stazinger, Jackson & Burd 2002) intermixes DFD with ER with UML syntax to create a curious jumping back and forth between modeling paradigms. There is no treatment of behavior-driven modeling or the use of polymorphism for problem structuring or analysis specification.

Each of these texts addressed a market and pedagogical standard set in the mid-1990's when the "object modeling wars" had only recently been settled with OMG 's blessing of UML. As UML and related methodologies are yet evolving, it is no surprise that the focus is almost universally on syntax alone. Table 1 indicates that no currently available text addresses behavior-driven modeling. In the following section we describe domain analysis including behavior-driven modeling based on metaphors and how they impact IS professional practice and reuse.

## 3. OBJECT-ORIENTED REUSE

The OO paradigm encompasses a broad range of concepts for describing, programming, constructing and managing information systems (Figure 1). Each development project adopting OO in one or more of these ways applies the paradigm to building and sustaining systems. Designers can consciously use OO to implant reusability into a development product. Specifically because of inheritance and polymorphism OO surpasses prior paradigms in its potential to support planned reuse.

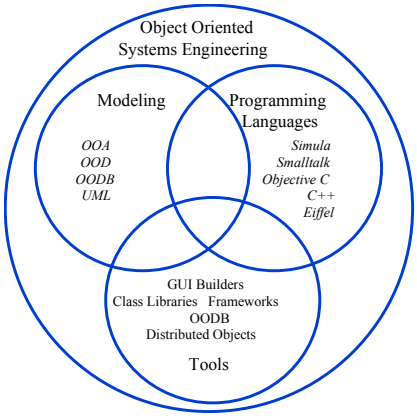


Figure 1: Facets of the Object-Oriented Paradigm

### Object-Oriented Tools

Software tools aid developers in manipulating abstractions at a higher level than lines of code or program fragments. Their design-

ers define a particular interpretation or subset of the modeling paradigm's abstractions to simplify or streamline the developer's tasks. Simplification exacts a price by either limiting expressive power or by requiring a high degree of conformance to the tool designer's interpretation of the modeling paradigm. Object oriented development tools are no different.

Object oriented tools capture the expressive power of OO to varying degrees of completeness and/or fidelity (Agarwal 1996). Because these tools are the expressions of OO that practitioners most frequently experience, they have formed the popular definition of object orientation.

**Exploiting OOSE for Reuse**

Each of the OO technologies depicted in Figure 1 enables reuse. OO modeling enables the evolution of description built upon specialization of fundamental system requirements. OO programming enables the reuse of software function either through replicating objects based upon class definitions or the derivation of new structures through subclasses. OO tools provide component integration and transformation along prescribed extension points defined in the architecture of a framework or tool kit. OOSE integrates the technologies and stewards the development and management of system knowledge in the class library. In every case, however the benefits of OO technologies depend completely upon the efficacy of the model of requirements (business rules and metaphors) and whether or not they can be effectively and efficiently represented and reused with the object tools. The next section addresses modeling to enable the benefits of object-oriented development.

## 4. DOMAIN MODELING

Information system modeling depends upon a specific requirement scope – the context within which an artifact is described. (See Figure 2.) Object orientation's effectiveness in reuse is critically sensitive to requirement scope.

**Requirement Scope**

Analysts and designers may consider a range of scopes when describing an information system artifact. We define three levels of scope for reference:

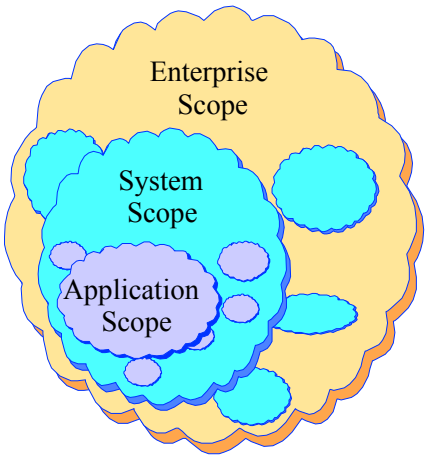


Figure 2: Requirement Scope

Application: the collection of information attributes and behaviors supporting a business function,

System: the collection of applications and their interrelationships that support a functional area within an enterprise, and

Enterprise: the collection of systems that encompass the business information and practices defining the operation of the enterprise or industry as a whole.

Most programming language based treatments of object identification and specification (Coad & Yourdon 1991, Coad 1992; Nerson, 1992; Rubin & Goldberg 1992) examine modeling within a single application. Although programming focuses almost exclusively on constructing individual applications, reuse depends on differences and similarities that can be detected and projected across applications and systems. Consider the contemplation appropriate when attempting to craft a class library that must support a family of applications. Inheritance and polymorphism structures must consider the next higher level of relevant requirement context to ensure that similarities and differences are grounded in domain level business rules rather than application level design choices. This entails understanding and managing *domain* issues with more complex interrelationships – *domain modeling*.

**Implications for IS Education**

Computing students infrequently encounter development exercises at the system or enterprise level. However, because requirement scope frames the questions of system efficiency and cost effectiveness, the same requirement or design question may require quite a different answer when posed in each of the three reference scopes described above.

To conduct domain modeling, the student must make explicit abstraction decisions – similarity and difference particularly with regard to inheritance and polymorphism. These decisions will have to span systems and eventually the enterprise. These decisions depend upon a qualitatively different collection of requirement and modeling questions than have been the tradition in IS education. Object-oriented technology cannot achieve cost-effectiveness without domain analysis.

Object oriented systems engineering and component based systems engineering are a driving force in IS development for the foreseeable future (Duggan 2002). They are based upon domain modeling using behavior-driven abstraction and the formulation of metaphors. Educating future systems professionals in domain modeling is at least as important as teaching OOP – if not more so. Educating students in domain modeling requires an emphasis on behavior-driven modeling.

## 5. BEHAVIOR-DRIVEN MODELING

> *"The greatest thing by far is to be a master of metaphor. It is the one thing that cannot be learnt from others; and it is also a sign of genius, since a good metaphor implies an intuitive perception of the similarity in dissimilar."* Aristotle

Modeling revolves around three views of system: functional, static and dynamic: Functional – user visible, Static – structure, and Dynamic – interaction of objects to accomplish the user visible functionality. In UML these views are accommodated by use cases, class / object diagrams, and by activity, sequence and interaction diagrams, respectively. The modeling process is one of discovering the essential characteristics of a system in these three dimensions and then constructing a system of metaphors that honors that essence absent unnecessary constraints. In behavior-driven modeling we refer to these metaphors as business rules. Focusing on business rules is a key to broad-based reuse.

**Business Rules**

We use *business rule* to mean an "architectural facet" of a behavior-driven model. Business rules are constraints that define the variety and range of variation of behaviors that are allowable among autonomously defined classes / objects. *(As definitions of what is "allowable," business rules shape a model's future evolution as well as its present structure and behavior.)* A business rule may be an "integrity constraint" as in entity-relationship models that determine the consistency of relational operations applied to defined entities. It may define a formula of computation or pre-/post-conditions of business actions. Business rules emanate from within the application, system or enterprise scope of a requirement being modeled. A business rule defined within a particular scope projects its interpretation throughout any subordinate scopes (i.e. a business rule defined within a system scope extends to any applications within that scope as well.). Variations on a business rule at a subordinate level (e.g. a subclass or method override) must conform to the business metaphor that the original rule prescribed. This conformity not only maintains the metaphor but reinforces it and enables polymorphism and a continuity of understanding among developers and users alike.

Contrary to a business rule, an *accident of implementation* denotes a behavior perceived in a system that is not defined by a business rule, but rather is the legacy of a designer's (or user's) choice of presenting or combining some set of otherwise legal system behaviors. Accidents of implementation commonly arise from choosing a particular implementation style or technology. Business rules seldom (if ever) arise from choices of batch vs. interactive or text vs. multi-media, for example. They more often arise (more insidiously) from the common practice of users formed by habit rather than business intention.

The upshot of this way of thinking is that achieving an "exact" object model of the current system's behavior (modeling the system with "accidents of implementation" in

tact) actually obscures the reality of the business rules. This does not lead to extensive reuse potential but often leads to an otherwise "process driven" model representation depicted in object symbols. Behavior-driven modeling results in object models that leave the "choices of implementation" up to the implementer constrained only by the business rules that define the business requirements.

**A Behavior-Driven Example**

As a simple example consider modeling the routine activity of reconciling a checkbook, matching one's personal records with those of the banking institution. By way of contrast consider the results of traditional process-driven and data-driven approaches before the results of a behavior-driven thought process.

Procedure-driven modeling focuses on the steps naturally evident in the current practice of reconciliation: entering transactions in the check register, receiving a monthly statement, and merging the entries of both to resolve any differences. This approach commonly results in a singled-threaded, sequential depiction of problem domain activities.

Data-driven modeling would more often focus on the questions that would need answers in the reconciliation process: "What information is recorded on a transaction record?" and "What information indicates the consistency of a register entry with the bank's information?" This approach commonly results in a collection of un-sequenced queries eventually to be organized in application interface design.

Behavior-driven modeling attempts to identify underlying business rules that define good behavior in the problem domain while at the same time affixing those rules to the tightest focus of responsibility possible: "How does a check know if it is reconciled?," "What actions does a transaction take to reach a reconciled state?" and "How are checks, credit card and debit card transactions the same and / or different?" This latter mode of modeling reveals that the practice of monthly statements is an accident of implementation in the banking system rather than a business rule defining reconciled transactions. It reveals that although checks, ATM and debit transactions may be governed by differing permissions, they share an identical core metaphor of transaction. These realizations free the implementer to consider real-time, wireless reconciliation as soon as a bank clears a transaction or including new transaction-based products without modifying the underlying domain model in any way.

Indeed, both procedure-driven modeling and data-driven modeling suffer from their heritage of sequential-thinking born of the "input-process-output" model of computing. Behavior-driven modeling accommodates the more realistic asynchronous interaction of business rules – an approach that does not relegate the consideration of rarely observed business rule combinations to the status of exceptions.

## 6. SUMMARY

This paper briefly surveyed the current state of OO modeling in IS2002 and commonly used systems analysis and design texts for IS2002.7. It summarized the OO paradigm's key role in achieving industrial-scale, enterprise-wide reuse. We conclude that the current level of treatment of domain modeling is not commensurate with the emphasis that it is receiving in professional practice today. We further examined the need for OO modeling with a focus on requirement scope to enable reuse opportunities.

We propose that to adequately prepare computing graduates for domain modeling a new emphasis on behavior-driven modeling incorporating system-wide and enterprise-wide analysis is needed. We have demonstrated the distinction between these approaches and the traditional process-driven and data-driven application focus that is prevalent in current curricula and accompanying texts. We conclude there is a clear need for a detailed review of IS2002 and associated national IS curricula models with a purpose of increasing the integration of behavior-driven and metaphor-based domain modeling in IS education.

## 7. REFERENCES

Agarwal, R., A.P. Sinha, and M. Tanniru (1996), "Cognitive Fit in Requirements Modeling: A Study of Object and Process Methodologies," Journal of Management Information Systems, 12(2): 137-162.

Arlow, J. & I. Neustadt (2002), UML and the Unified Process, Practical Object-Oriented Analysis & Design, London, GB: Pearson Education Limited.

Brooks, F. P. (1987), "No Silver Bullet: Essence and Accidents of Software Engineering," IEEE Computer (April), 10-18.

Coad, P. (1992), "Object-Oriented Patterns," Communications of the ACM, 25(9), 152-159.

Coad, P., & E. Yourdon (1991). Object Oriented Analysis (2nd Edition ed.). Englewood Cliffs, NJ: Prentice-Hall.

Coad, P., & Yourdon, E. (1992). Object-Oriented Design. Englewood Cliffs, New Jersey: Prentice-Hall.

Dennis, A., B. Wixom & E. Tegarden, (2005). Systems Analysis and Design with UML Verson 2.0 An Object-Oriented Approach (2nd Ed.), Hoboken, NJ: Wiley.

Duggan, J. (2002), "Successfully Selecting Object-Oriented A&D Tools," Gartner Group.

Gorgone, John T., Gordon B. Davis, Joseph S. Valacich, Heikki Topi, David L. Feinstein, and Herbert E. Longenecker, Jr. (2002). Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS), Association of Information Technology Professionals (AITP).

Harris, D., (2003), Systems Analysis and Design For Small Enterprises (3rd Ed.), Boston, MA: Course Technology.

Hoffer, J., J. George & J. Valacich, (2002). Modern Systems Analysis & Design (3rd Ed.), Upper Saddle River, NJ: Pearson Education, Inc.

Hoffer, J., J. George & J. Valacich, (2005). Modern Systems Analysis & Design (4th Ed.), Upper Saddle River, NJ: Pearson Education, Inc.

Jacobson, O., M. Griss & P. Jonsson (1997). Software Reuse. New York, NY: ACM Press.

Nerson, J.M. (1992). "Applying Object-Oriented Analysis and Design." Communications of the ACM, 35(9), 63-74.

Rubin, K. S. & A. Goldberg (1992). "Object Behavior Analysis." Communications of the ACM, 35(9), 48-62.

Schach, S., (2004). Introduction to Object-Oriented Analysis and Design With UML and the Unified Process, New York, NY: McGraw-Hill/Irwin.

Satzinger, J., R. Jackson & S. Burd (2002). Systems Analysis and Design in a Changing World (2nd Ed.), Boston, MA: Course Technology.

Stumpf, R. & L. Teague (2005). Object-Oriented Systems Analysis and Design with UML, Upper Saddle River, NJ: Pearson Education, Inc.

Waguespack, Leslie J., Jr. (1994), "Domain Analysis is an Essential Skill of the OO Analyst," Proceedings of ISECON '94, Louisville, KY, October.