ISSN: 1545-679X

Information Systems Education Journal

Volume 5, Number 20 http://isedj.org/5/20/ June 6, 2007

In this issue:

A Perspective on the Use of Modeling Diagrams in Computer Science and Information Systems Curricula

David R. Naugler
Southeast Missouri State University
Cape Girardeau, MO 63701 USA

Ken Surendran
Southeast Missouri State University
Cape Girardeau, MO 63701 USA

Abstract: Modeling diagrams are used in Computer Science and Information Systems courses. Different tools are used for different paradigms of system development. The authors share their perspectives in using different modeling tools in systems analysis and design and database courses. They discuss paradigm related issues in programming languages. They suggest using the diagrams from both the paradigms (procedure centric and object oriented) with a view to enhancing the value of the curricula.

Keywords: modeling tools, analysis and design, system development paradigms

---

Recommended Citation: Naugler and Surendran (2007). A Perspective on the Use of Modeling Diagrams in Computer Science and Information Systems Curricula. *Information Systems Education Journal, 5* (20). http://isedj.org/5/20/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2005:* §3364. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/5/20/**

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.

A Perspective on the Use of Modeling Diagrams in Computer Science and Information Systems Curricula

David R. Naugler
dnaugler@semo.edu

Ken Surendran
ksurendran@semo.edu

Southeast Missouri State University
Cape Girardeau, Missouri 63701 USA

ABSTRACT

Modeling diagrams are used in Computer Science and Information Systems courses. Different tools are used for different paradigms of system development. The authors share their perspectives in using different modeling tools in systems analysis and design and database courses. They discuss paradigm related issues in programming languages. They suggest using the diagrams from both the paradigms (procedure centric and object oriented) with a view to enhancing the value of the curricula.

Keywords: Modeling tools, analysis and design, system development paradigms

1. INTRODUCTION

The term tool, as used in this paper, is not necessarily a software product and may refer to standards as well. Many important design tools are notations or collections of concepts that do not necessarily have or need a software implementation. Such tools can be considered separately from any implementation. Thus Unified Modeling Language (UML), a graphical modeling language, is considered to be a modeling tool, as are design patterns, Entity Relationship Diagrams (ERDs) and normal forms.

Virtually all software explicitly or implicitly involves modeling in some form during its design and construction. At all stages in the design and construction of software, models are used, often implicitly, to guide development. The whole system development process consists of a series of transformations starting from abstracting the system in the problem domain and realizing solution in the computer domain. During this process, several intermediary system artifacts are produced in the analysis and design workflows using diagramming tools, before program construction. These intermediary models normally use standard diagramming languages for facilitating communication among the various participants in the development process.

However, a model is not the same thing as what it is modeling. Even the standard (IEEE754) floating point numbers used in most programming languages to model real numbers violate simple algebraic rules such as the commutativity of addition [Goldberg, 1991]. The difference between the model and the "reality" it abstracts is a factor in all modeling and in the conclusions that can be reached from using a system based on a model. The main problem with using models is abstracting the essence of what is being modeled so that solutions in the model correspond to real solutions of the problem. This applies at all levels of modeling.

Modeling diagrams are used extensively in the systems analysis and design course and, to some extent, in the database course. Once an information system is planned, the

actual construction is preceded by analysis and design during which the intended system is abstracted and blueprints for implementation are prepared. In view of the complex nature of analysis and design and its importance in system development, the IS2002 model curriculum (Gorgone, 2003) recommends three analysis and design courses. In the first course analysis and logical design are considered and in the remaining two, physical design and implementation issues are considered for applications in different environments. Even though the database course is not explicitly mentioned, it is embedded in one of the physical design courses.

The computing sciences use sophisticated techniques to model a software project. Large projects require such models. Smaller projects may not require such explicit models but may benefit substantially if such techniques are used. Modeling can be performed at a variety of levels. A model can be as simple as a few sketches on paper or so complicated that many thousands of pages of carefully written information is required to express it. Software Engineering explicitly studies the use of certain modeling methods for the purpose of designing large software projects. In Computer Science (CS) curricula, analysis and design topics are covered in software engineering courses. Also, many CS programs include a database course in their core curricula.

At the authors' university, the MIS program has both analysis and design and database management as core courses and the CS and CIS (Computer Information Systems) programs have software engineering and database as core courses. The authors teach modeling diagrams in procedure centric (PC) paradigm in the analysis and design course and UML diagrams in object oriented (OO) paradigm in software engineering courses. In the database courses, modeling diagrams pertaining to relational databases are taught. Having experienced the paradigm changes in these areas over the years, they observe a few similarities and differences between the models used in these paradigms. Based on these observations, they suggest a few possibilities in using some of the ideas from the OO paradigm in PC SA&D. Also, they highlight the difficulties faced in teaching OO paradigm along with relational databases in software engineering. In the next section, they briefly describe the two major paradigms used in systems analysis and design.

2. ANALYSIS AND DESIGN PARADIGMS

There are currently two major paradigms for analysis and design which reflect two somewhat distinct ways of solution conceptualization. Structured Analysis and Structured Design (procedure centric paradigm) and/or Object Oriented Analysis and Design (object oriented paradigm) are taught in Systems Analysis and Design (SA&D) courses. Of the two analysis and design paradigms, the procedure centric paradigm has been in use for quite sometime, whereas the object oriented (OO) paradigm has gained prominence relatively recently (since 1997). A series of innovations such as structured design (Yourdon & Constantin, 1979), the relational model for database (Codd, 1970) and the entity-relationship model (Chen, 1976) provided a basis for a formal procedure centric SA&D course. A large majority of the Information Systems programs continue to use a procedure centric approach in their SA&D course. However, after the introduction of UML in 1997 (Booch, et al, 1999) as "the" standard modeling language for the object oriented paradigm, more and more instructors are considering the object oriented paradigm for their SA&D course. For the sake of completeness, these two SA&D paradigms are very briefly summarized in the following.

2.1 The Procedure Centric Paradigm

Data and processes are considered separately in the procedure centric approach. The first model in a SA&D course taught using this paradigm is the context diagram. This is followed by several levels of data flow diagrams (DFDs). The separation of process and data and the focus on process is apparent in them. Concurrently, a corresponding entity relationship diagram is introduced to deal with the data. The initial logical diagrams are then transformed into physical ones that address architectural concerns. As part of detailed design, structure charts and schema are discussed. This brief description of the procedure centric paradigm considers a few main diagrams discussed later. User interface design, test plans, and implementation issues are also discussed.

Main SDLC steps	Paradigms	
	Procedure Centric Models	Object Oriented Models
Analysis	Context, Data Flow, Entity Relationship	Unified Modeling Language (Use case, collaboration, class, package)
Design	Structure chart, schema	UML (Sequence, statechart, object, class, subsystem, deployment)
Implementation	Procedure centric language	OO Languages

Table 1

Research and development in database technology has flourished relatively independently of this paradigm. Most present day applications use a relational database management system (RDBMS), although a few legacy hierarchical and probably also network databases are still in use. All types of databases are modeled well in the procedure centric approach since databases are designed independently of the processes.

2.2 Object Oriented Paradigm

Structured Analysis and Structured Design helps produce specifications suitable for implementing applications with process (procedure) oriented languages such as COBOL and encourages procedural programming in more so-called object-oriented languages such as C++, and Java. Even though object oriented languages have long existed, matching modeling standards for analysis and design were finalized only in 1997when the Object Management Group released Unified Modeling Language (UML) as the standard modeling language for expressing the analysis and design artifacts under OO paradigm. In a way, the lack of a suitable modeling tool limited the growth of OO applications. Table-1 summarizes the basic models under the two paradigms for the three main primary (system development Life Cycle) SDLC steps.

The inherent invisibility of software which makes system development difficult is addressed by providing five different views of the system under development: the use view, logical view, process view, implementation view and deployment view (Kruchten, 1995). UML diagrams can be used for depicting these views.

One approach to using the various UML diagrams in an SA&D course is briefly described here. Analysis and design using OO paradigm starts with a use case diagram and use case descriptions. (The use case model includes, in addition, supplementary quality of service requirements.) Using the use case descriptions, three groups of analysis classes are identified which collectively take on the responsibilities of providing the required services. Interaction (collaboration / sequence) diagrams for the scenarios of the use cases help in the above class identification activity. The non-functional requirements from the use case model help identify the analysis mechanisms some of the identified classes may require. These classes are suitably packaged paving the way for architectural design using package diagrams. The classes' analysis mechanisms are mapped into design mechanisms. In particular, the persistent entity classes (from analysis) become candidates for database consideration. The boundary classes (from analysis) are transformed into user interfaces. Required subsystems and their interfaces are identified. The possibility of using design patterns and frameworks are also examined. The behavior of complex objects is expressed using statechart diagrams. Finally, all design class diagrams are prepared.

In such an SA&D course, user interface and database design are also discussed. Both OO and PC SA&D courses use more or less the same contents for user interface design. However, the discussions on database design vary. Both the class diagram (which also contains the entity classes) and the entity relationship diagram (ERD) serve as diagrams for database design. The database courses use ERD as RDBMS are popular (and of practical value). In SA&D courses, relational modeling diagrams are still taught for the database aspects. Even in SA&D courses that deal with object oriented paradigm, the emphasis is placed on the relational modeling with an introduction to the designs dealing with OODMBS and ORDBMS (Satzinger, et al, 2004).

2.3 Relational Database

Relational database technology was a highly significant development in Computer Science

and is now a solid, stable and mature technology. Commercially available and widely used RDBMS's such as Oracle provide a very high level of dependability, security and support. Given the very considerable financial and intellectual investments in RDBMSs and their remarkable success - almost all non-toy programs use a commercial RDMS - users are reluctant to seriously consider alternatives and vendors are reluctant to make changes that break existing relational databases or that fail to maintain the current levels of dependability and security.

2.4 Relational to Object Oriented

The relational *model* handles objects with no problem. In fact, the model really does not define the data types that can be used. The atomicity requirement called the first normal form was adopted for practical efficiency and implementation reasons because of the early emphasis on model implementations and not on the model itself and is only artificially part of the commonly received model (Date, 2001, Fagen 1981). Implementations of the relation model – relational DBMS's - have built in data types and often only built-in data types.

Major RDBS vendors have moved slowly to incorporate object orientation into their products. Indeed, there are some very real and subtle difficulties and issues in the full incorporation of object orientation in DBMS which computer scientists have yet to completely solve. So-called object-relational DBMS are a transitional phase incorporating some object concepts while maintaining the security and dependability of relational systems. Intersystems' Caché database system, which uses what the company calls the postrelational database model, is an established commercial product which can reasonably be called object oriented and which interfaces well with some object oriented languages in common use (Kirsten et al, 2003).

2.5 Use of Object Oriented Databases

Most developers and users are reluctant to consider pure Object Oriented DBMSs (OODBMS). Indeed, many object-oriented languages and platforms have strong support for relational databases. Both Java and the Visual Studio languages (most importantly C#, and Visual Basic) provide much support for interacting with relational databases through SQL and even creating and manipulating relational databases in memory. As a result most developers on such platforms think of databases as relational databases and have developed the knowledge and skill to use them. The Caché database system certainly allows interaction between the database and languages such as Java and C++ but Oracle and SQL Server seem to be dominant currently.

Jade, a product from New Zealand, offers a consistent two-in-one OO development environment in that it is an OO language with OO database (Jade, 2005). Such an integrated product could be used in capstone courses involving the development of a new software entirely (analysis, design, implementation – including database) in the OO paradigm without having to use any DB connectivity tools.

3. PROGRAMMING LANGUAGES AND SYSTEM DEVELOPMENT PARADIGMS

The types of programming languages available for system implementation affect the selection of system development paradigm. Availability depends on the existence of effective compilers or interpreters, language specific or compatible programming support tools such as development environments, the knowledge and skill of the programmers, and the support of management. Perhaps the most important consideration is the programming paradigms with which the implementers are comfortable.

3.1 Language Paradigms

A programming language is fundamentally a tool used by programmers to express algorithms. Different languages have different syntax and may provide the programmer with different constructs. Different programming language paradigms provide distinct ways to conceptualize algorithms, and hence distinct ways to think about problems. An important "bonus" of most programming languages is that they are implemented so the algorithms expressed in them can be "automatically" compiled to produce code that actually runs. Programmers can easily continue to think and program procedurally in any programming language, although this is more difficult in some than in others.

3.2 Object Oriented Languages

Programming languages may be categorized into paradigms in various ways. Most useful ways do not lead to disjoint categories. Object orientation is way of conceptualization which leads to many quite different appearing instantiations in programming languages when added to or used in conjunction with other paradigms. One of the earliest object oriented languages was Simula 67 which was built on top of Algol 60 in much the same way as C++ is built on C. Algol 60, the first programming language that was carefully designed, is the direct or indirect ancestor of most procedure oriented languages. Some form of object orientation has been added to many procedure oriented languages. The languages most commonly used for commercial program construction, such as C++, Java and more recently C# and VB.NET, are of this type.

Most programming using so-called object oriented languages is procedural programming, at best using the syntax the language provides for objects. It is important to remember that object-oriented programming is a programming paradigm which is not synonymous with using the syntax of classes/objects in object oriented languages.

In Information Systems we are concerned with object oriented modeling primarily for the construction of systems using object oriented tools in what may be considered the usual languages such as C++ and Java. In such languages classes are a very heavy duty construct carrying a great deal of the load of the design and construction of programs. Not to be left out of consideration, modern COBOL versions have classes. Other, less widely known and used, languages such as SML and Unicon have powerful constructs lacking in C++ and Java. In these languages classes maybe available but serve a somewhat minor function since much of the power of the languages comes from other constructs. Neither of these languages needs object orientation for effective large scale programming. [Note: SML does not itself have classes – the variant O'CAML is essentially SML with classes. There are some subtle issues with adding classes while maintaining SML's typing systems]. On the other hand, the language Smalltalk (Kay, 1993), is often called a pure object oriented language since virtually everything, including control constructs and literals are objects.

It can be observed from the above discussions that concepts from new paradigms were incorporated in older procedure-centric languages to enrich them. In the same vein the useful concepts in OO analysis and design could be used to enrich the PC SA&D. These suggestions are indicated in the next section.

4. OBSERVATIONS AND SUGGESTIONS

The Unified Modeling Language (UML) has become a very important modeling tool for software projects. It is the confluence of several major approaches to object oriented analysis and design (OOAD) models in software engineering. Procedure centric modeling tools predate UML and could benefit by using OOAD heuristics.

Use cases are not limited to the OO paradigm and can enhance learning in a course using the PC paradigm. They can be used to verify analysis and design artifacts ensuring they are in sync with earlier requirements. This is not achieved with context diagrams. Use case diagrams also provide context information and serve some of the functions of level-0 DFD minus the data-stores. This needs to be further examined. Some textbooks (Dennis & Wixom, 2003) introduce use cases just before process modeling.

Use case descriptions are helpful in user interface design.

Activity diagrams, like use cases, need not be restricted to OO. Capturing business processes usually precedes requirements analysis.

During use case analysis entity classes are identified and their analysis mechanisms are defined. Analysis classes with persistence mechanisms are candidates for ERD. Heuristics used for identifying entity classes can also be used. Collaboration diagrams drawn from use case scenarios can provide considerable insight for the preparation of ER diagrams, particularly in finding the related entities and relationship types.

Teaching OO SA&D is easier when students have a considerable background in object oriented programming. However, nearly all students find it difficult when a relational database is chosen for handling persistent

classes. This requires additional redesign from OODBMS to RDBMS. Class diagrams are much more complex than ERDs since they abstract complex relationships not found in ERDs. Class diagrams are suitable for OODBMS which are not yet in common use. In some higher level database courses mappings between object and relational database designs are considered (Dietrich & Durban, 2005).

The perspectives presented in this paper are from the academic trenches. It would be interesting to know the trends in the industry concerning the paradigm uses and paradigm mixes.

5. REFERENCES

Booch G., J. Rumbaugh, I. Jacobson (1999). *The Unified Modeling Language User Guide*, Addison Wesley, 1999

Date, C.J. (2001), *The Relational Model: A Retrospective Review and Analysis*, Addison Wesley, 2001

Dennis, A. and B. H. Wixom, (2003), *Systems Analysis and Design*, 2nd edition, John Wiley & Sons

Dietrich S. W. and S. Urban (2005). *An Advanced Course in Database Systems: Beyond Relational Databases*, Pearson Prentice Hall.

Fowler, Martin (1997), *UML Distilled: Applying the Standard Object Modeling Language*, Addison Wesley, 1997

Goldberg, David (1991), "What every computer scientist should know about floating-point arithmetic", ACM Computing Surveys, Volume 23 Issue 1 March 1991

Haugland, S., M. Cade and A. Orapallo (2004) *J2EE 1.4 The Big Picture*, Prentice Hall

Jade, (2005) Retrieved Sep 21, 2005 from. http://www.jadeworld.com/education /jadetep.htm

Kay A. (1993). "The Early History of Smalltalk", ACM SIGPLAN Notices, Volume 28, No. 3, 1993, pp 2-54

Kirsten W, M. Ihringer, M. Kühn and B. Rohrig (2003) *Object-Oriented Application Development Using the Caché Postrelational Database*, 2nd Edition, Springer Verlag

Kruchten, P., (1995). "The 4+1 View Model of Architecture," IEEE Software 12(6).

Satzinger, J. W., Jackson, R. B., Burd, S. D., (2004). Systems Analysis and Design in a Changing World, 3rd Edition, Thomson Currier technology.