Introducing the Unified Modeling Language into the Information Systems Curriculum

Donald Golden
Victor Matos
Department of Computer and Information Science
Cleveland State University
Cleveland, Ohio 44115, United States
golden@cis.csuohio.edu matos@cis.csuohio.edu

ABSTRACT

The Information Systems undergraduate program at Cleveland State University (CSU) has adopted the Unified Modeling Language (UML) as the tool for defining a system model through out the development process. This paper discusses the revised CSU curriculum, the use of UML as a common tool to unify several aspects of system development and implementation, and the way in which this approach replaces traditional software development tools. In general, the current CSU curriculum matches the IS 2002 curriculum model in which the implementation of database systems is a continuation of the analysis and logical design course. At CSU, to simplify course scheduling issues, students may take either course first or both at the same time, compromising the IS 2002 suggested prerequisite structure. Our teaching approach to this academic scenario is presented, as is our teaching methodology. Several examples are shown, including the use of tools such as Rational Rose, MS-Access, and Oracle.

Keywords: IS Curriculum Design, Database Implementation, UML, ERD, Systems Analysis, Object-Oriented Systems

1. INTRODUCTION

In 1998 the Information Systems curriculum at Cleveland State University (CSU) underwent a major renovation, based in large part on the IS '97 (Davis, 1997) model curriculum, and has recently undergone review based on the IS 2002 report (Gorgone et al., 2002). Our goal at CSU is to prepare our students to play an effective role in entry level information systems positions, and to give them a solid foundation that allows them continued career growth. The flexibility of the IS 2002 model curriculum fits well with that goal. The current structure of the CSU IS curriculum is shown in Figure 1; the diagram also shows how each of our courses compares with the corresponding IS 2002.X course model; for instance our Systems Analysis Methods (IST 321) closely matches the IS 2002.7 course suggested in (Gorgone et al., 2002).

In addition to business courses and general education courses, the Information Systems program requires students to take a core of eight computer-based courses and three elective courses. The core covers programming, systems analysis, database development, networks, and the relationship between information systems and business. Electives include subjects such as web site development, knowledge management, project management, advanced programming, emerging technologies, etc.

In addition to the fact that the CSU program does not include quite as many courses as the IS 2002 program, there are differences in course prerequisites, particularly those relating to the systems analysis and database implementation courses. Many of our students (possibly a majority) are part-time students, which means that they take a few courses each semester and cannot fit into any type of "lock-step" program. As a result, we try to avoid locking students into long chains of course prerequisites since doing so could extend their degree completion time by a year or more. One consequence of this situation is that for many of the courses beyond the first few semesters, we cannot control the order in which students take courses. In particular, we cannot predict whether students will take the systems analysis course before the database course or vice versa. Given this situation, we have structured each course so that, in addition to teaching about the courses main topic, we also show students how the topic fits into the system development process.

In the initial redesign of the program (circa Fall, 1998), we identified two goals to emphasize across the curriculum: (a) placing emphasis on problem solving strategies and critical thinking; and (b) identifying and using best-practice and state-of-the-art techniques in problem solving. At that time, our general approach to system development was structured, and except for obvious knowledge dependencies, we treated most courses as independent of each other.

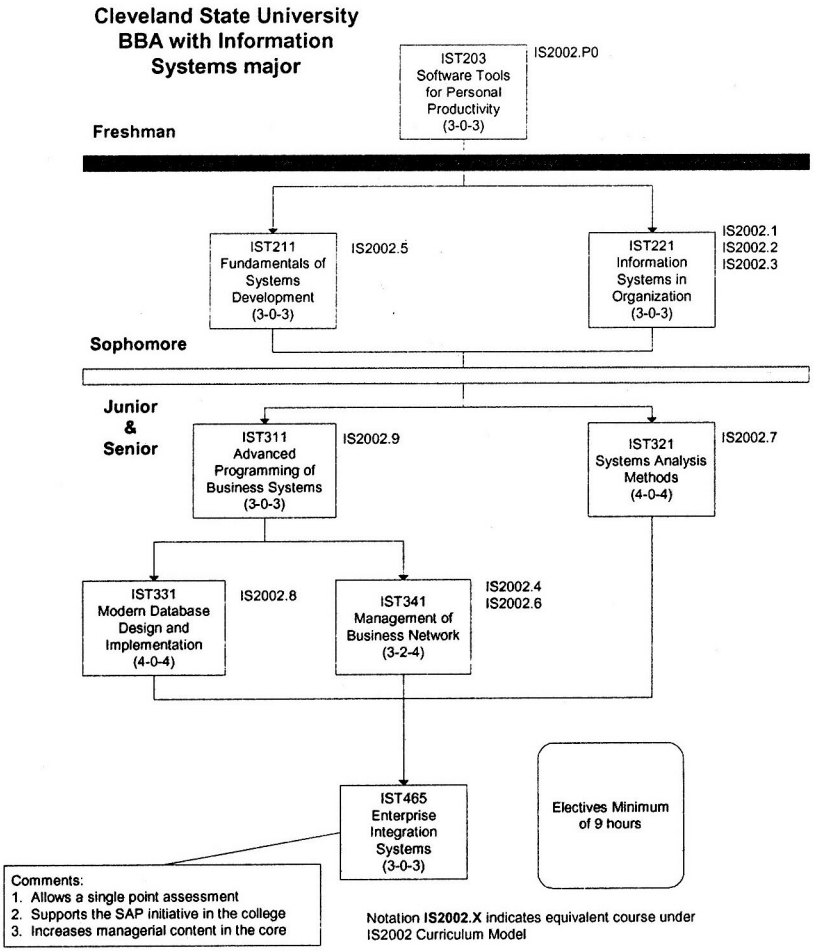

Figure 1 – Current IS Course Sequence

Over the next few years (the early 2000's) it became clear that we should be teaching object-oriented systems development and several changes were made to the curriculum. First we adopted a third goal: (c) presenting the entire systems development effort in a common, uniform, and unambiguous framework. Today, IS courses embrace the object-oriented model, and traditional structured development tools such as entity-relationship diagrams, data flow diagrams, and structure charts (Ramakrishnan and Gehrke, 1998; Yourdon, 1989; Page-Jones, 1988) have been replaced by UML diagrams. This has proven to be highly beneficial for several reasons. The UML tool provides a single complete and extendable tool for representing the entire life cycle. Although mastering the whole range of UML diagrams may seem overwhelming for the novice student, by presenting different parts of UML in different courses we make the learning process manageable, and students respond well to learning about specific types of diagrams. More importantly, the repetition of the diagrams in different IS courses has a remarkable effect in helping students to understand, review, and accept the systems development effort as a continuous and complex process. Rather than associating UML diagrams with just analysis and design, they are exposed repeatedly to the same tool in different parts of the system life cycle.

As a result, faculty have been willing to adopt UML as the common language to depict a problem as well as its solutions, and UML has proven to be appropriate for instruction at several levels. Different courses use UML diagrams that are appropriate to their specific needs. For instance, the intermediate programming course (IST 311) extensively presents use case, class, sequence, and state diagrams as ways of describing problems and their solutions using object-oriented tools. The database course (IST 331) has adopted the UML Data Model Profile (Gornik, 2003) as the mechanism to represent the implementation effort. Networking courses use deployment and activity diagrams to describe the hardware configuration and transactional behavior of a network, and so on.

2. ENTITY RELATION DIAGRAMS AND THE UNIFIED MODELING LANGUAGE

There is no question but that Entity Relation Diagrams are a perfectly satisfactory tool for developing databases. After

all, they have been used extensively in database design since 1976, when they were first introduced by Peter Chen (Chen, 1976); and today many commercial ERD software tools are available for both database developers and students, including products such as Erwin[1], Sybase DataArchitect[2], Oracle Modeling[3], Visible Analyst[4], etc.

A strong argument in favor of the use of ERDs is their minimality, simplicity, and ability to document related data issues that are important to the database developer from a structural viewpoint. ERDs support the illusion that a complete and reliable data organization could support any type of future operations. To draw an analogy with the construction industry, having a well-designed and solid foundation allows the builder to erect many kinds of structures. Extrapolating this statement might suggest that any kind of database could be implemented on top of a solid ERD, but this is not necessarily true. A weakness of the ERD model is its inherent lack of support for a behavioral description of entities. I.e., what do entities do, in what activities do they participate, when do events happen, what is needed for behaviors to work, what result is produced, etc.

In typical structured analysis and design courses, a collage of heterogeneous and non-integrated diagrams must be used to document entities' behavior (Yourdon, 1989). Indeed, in 1998, Paul Dorsey published an on-line article (Dorsey, 1998) which not only pointed out the problems of documenting behavior using ER diagrams, but which concluded, "ERD's [sic] are dead. There is absolutely no reason that people should still use a design method, which, in almost every way, has been improved upon by UML".

Even earlier, some studies (Sharble & Cohen 1993) seemed to suggest empirically that the data-centric design approach supported by ERDs was less effective than the function-centric view of object-oriented models. In an experiment they conducted, the Function-Centric solution resulted in a set of classes far more reusable and with behavior more evenly distributed than equivalent solutions based on the data-centric ERD solutions.

Furthermore, ERDs see little or no use outside the database field. This caused no problem in the past since the design and implementation of databases was generally viewed as an independent process with little connection to other system development activities. However, as information systems become larger and more complex, it has become increasingly important to treat the development process as an integrated whole, from the discovery and documentation of user requirements and business rules, to the creation of a logical system model, to the design and implementation of the physical system. With the advent and increasing use of object-oriented systems and methodologies, this development sequence becomes much smoother and produces a system which is easier to maintain and modify.

While the design and implementation of the physical database is still a specialized field, the initial specification of the database – the discovery of data requirements, the creation of a logical data model, etc. – becomes an integral part of the system development process. The Unified Modeling Language (UML 2.0) has become the de facto standard for object-oriented system specification, and its class diagrams have all the data specification capabilities of ERDs. For this reason alone – the fact that one tool can serve the needs of many phases of the implementation process – we believe that the value of using of ERDs in database courses has decreased substantially. If class diagrams can serve the same purposes as ERDs and are used in other parts of the system planning process as well, why should students be asked to learn a second, redundant tool?

In addition to the redundancy of ERDs as a development tool, we view the database design process itself as an extension of the overall system development process, an approach that reflects the recommended curriculum proposed in the IS 2002 report. This is not to say that database design has become part of systems analysis. Rather, we view database design as a branch coming off an initial system modeling root, just as process implementation, data communication, and other specialized components of the overall implementation process are branches coming off the same root.

3. PEDAGOGY APPLIED TO THE TEACHING OF SYSTEMS ANALYSIS AND DATABASE

Adopting UML as a common theme across the IS curriculum provides the students with a framework for continuous and consistent learning. Our experience using UML as a bridge between systems analysis and database implementation has been very positive, and the next two sections provide an overview of the academic strategy used at CSU to deliver the systems analysis and database courses. Section 3.1 summarizes the approach we use to guide the student during the creative process of design in order to produce an object-oriented depiction of a system. To make the issues more concrete, we use a Borrower-Library case. Section 3.2 briefly describes the database course using the Library case to illustrate how the UML database model profile could be used to convert the early conceptual specifications into representations appropriate for the physical implementation of a relational database.

3.1 The Systems Analysis Course (IST 321)

The systems analysis course begins with a discussion of the overall concept of planned system development, followed by an introduction to the system development life cycle. Overall, the methodology that is used is based on the Rational Unified Process (RUP) (Kroll and Kruchten, 2003), but we use the process as a guide, not a straightjacket.

Requirements specification is the first life cycle activity discussed in depth. We utilize use cases extensively, with emphasis on the interaction between the user and the system. Students are asked to avoid (or at least minimize) any aspect of physical implementation at this stage. On the other hand, we emphasize that every action performed by the system must be triggered by some event, either a state change within the system (e.g., an attribute reaches a critical value) or an action performed by a user.

Although there is no attempt in systems analysis to discuss anything about how data is structured or stored, students are taught to be specific about what data is visible when a task is executed. For example, it is perfectly correct to say that the system shows the customer's name, address, and payment history (date and amount of each payment) for the past six months, without being concerned (yet) about whether the data is actually stored in the system, how it is structured, etc. However, it is not acceptable to say that "the system displays the customer data," since this statement is too fuzzy.

Once students have learned about use cases, the next step is to develop classes. We use two steps for this activity. Students begin with the "noun analysis" process commonly used with RUP. That is, they identify the nouns in use cases, establish a list of candidate classes, and then eliminate the candidates that are not appropriate for classes. This activity creates a working set of classes, but those classes are little more than names. In the real world there are many ways to add flesh to these skeletal classes, depending on issues such as the analysts' familiarity with the system domain, the nature of the system, its relationship to existing systems, etc. To simplify the process for the classroom, we add basic attributes to classes by using the students' general knowledge of the domain (e.g., a university registration system), then move into the CRC (Class/Responsibility/-Collaboration) process (Wilkinson, 1995) to refine the set of attributes in each class and to start adding functionality to the classes. Although the CRC process is not actually a part of the Rational Process, it is commonly used as part of object-oriented analysis and we find it to be an effective way for students to learn how to determine class functionality and object interactions.

Initially, the behaviors (methods) assigned to classes are only brief descriptions such as "Compute Grade Point Average." The next step, therefore, is to define both methods and tasks (e.g., "Produce Grade Report") in greater detail, including documenting class relationships and object interactions or collaborations. Although we start using class diagrams as soon as students understand the general concept of classes, at this time we refine the class relationships with concepts such as aggregation, composition, and inheritance, using IBM Rational Rose[5] as the modeling tool. We also introduce control and boundary classes, start to refine the behaviors that are assigned to classes, and use sequence diagrams to show the collaborations between objects that support more complex tasks.

Next, we need to consider not only what behaviors and attributes must exist, but the class to which each attribute or behavior is assigned. For example, consider a decision that must be made in the library example used by Wilkinson (Wilkinson, 1995, p. 48). In this case study, borrowers (people using the library) borrow books and other items from the library. Before a borrower can check out another book, the system must verify that none of the items already on loan to this borrower are overdue. This raises the question of which class has the responsibility for determining a borrower's eligibility to obtain additional books from the library. To answer the question of eligibility, we need to know which books the borrower already has on loan and whether any of them are overdue, which in turn means that we need to know when each book is due to be returned.

The way in which we allocate the behaviors needed to answer these questions affects how certain attributes are allocated to classes. For example, do we want to make a list of due dates be an attribute of the Borrower[6] class? Experience using this example in class indicates that this is, in fact, how most students would address the problem. That is, give the Borrower the responsibility for determining eligibility, and give Borrower all the attributes needed to make the decision. However, this approach ignores the fact that determining if a borrower's book is overdue as part of determining eligibility is simply a special case of determining if a book is ever overdue, which suggests that the responsibility for determining if a book is overdue belongs to the Book class, and that Borrower needs to collaborate with Book to determine eligibility. In this case, although Borrowers need to know which books they have on loan, only Books need to know their due date.

Of course, in the analysis phase of the life cycle we are not concerned with physical data requirements. During database design, the designer may decide to put a copy of the Book's due date with the Borrower data for the sake of efficiency, but that type of decision is irrelevant at this stage. Nonetheless, most students come to a systems analysis course from one or more programming classes, and they find it difficult to ignore physical data considerations. We try to discourage this type of thought, but since it seems to occur naturally we also seek to emphasize good data design techniques, always with the emphatic statement that the students will learn more about the topic in the database course. For those students who have already taken the database course, the emphasis is on the fact that they are not designing databases, and that they should focus on logical considerations, not physical ones.

There are, however, techniques used by relational database designers that we believe are of value to the analyst. In particular, we feel that simple normalization (through third normal form at most) can give useful insights into a system's structure. For example, consider the commonly used example of a customer purchase order. This business document typically consists of header information such as the date, customer name and address, etc; a body, containing of one or more line items, each of which typically consists of a quantity, item description, unit cost, and extended cost; and a footer, which typically will consist of items such as a subtotal, tax, shipping or handling charges, and an order total. Although we can certainly create a Customer Purchase Order class that contains all this information, doing so hides the fact that the source document really contains data relating to customers, data relating to products that our company sells, and data that pertains only to this particular purchase. While it is not necessary at this time to be concerned about details of data storage, it is useful to the analyst to be aware of this Customer-Product-Purchase relationship since this awareness can help to understand relationships with other

services in the system (such as inventory management, purchasing, accounting, etc.).

Finally, we should emphasize that the process of defining data and behaviors for classes also has the effect of defining the ways in which those behaviors constrain the data. I e., in defining class behaviors, we also define (and document) the business rules that must later be implemented as data constraints, triggers, etc.

In summary, although IST 321 places no emphasis on physical data structuring (and, indeed, discourages physical considerations), there is a conceptual relationship between systems analysis activities and database implementation. Systems analysis is the starting point for understanding what data is inherent in the system domain, what logical structures exist in the data, how attributes should be assigned to classes to maximize the logic and flexibility of the system, how the data must be manipulated to provide the services that users require from the system, which classes perform the manipulation, and what relationships exist between those classes. For those students who have not yet studied database design, we provide the logical foundation on which a physical relational database should be built. For those students who already have studied database design (or who are taking the database course concurrently with the systems analysis course), we show how the logical model is created and how to understand the language that defines it.

3.2 The Database Design Course (IST 331)

Currently our students are prepared to interact with Relational DBMS technology (RDBMS). However plans have been made to include in the near future tools emerging for the Post-Relational approach, such as Caché[7]. Clearly the object-oriented nature of a Post-Relational DBMS environment will favor UML depictions of the structural and behavioral characterization of the system to be dealt with.

For the relational database developer, producing an appropriate software solution includes consideration of many factors such as the design of tables, evaluation of the physical organization of the database (possibly over a network and multiple computers); storage of the data (perhaps spanned over multiple disks located in multiple locations); efficient access to the data; implementation of the various views required by the different parts of the system; implementation of business rules expressed in the form of triggers and constraints; construction of the presentation layer; migration of legacy data; design of security mechanism; recovery plans for non-catastrophic failures; daily operation and administration of the data resources, etc. To cover these subjects, the database syllabus includes a broad range of topics such as data models and modeling tools, SQL, programming tools for the database developer, etc.

At CSU two types of RDBMS platforms are discussed: Microsoft Access for small applications, and Oracle for enterprise scale solutions. Emphasis is given to the mastery of SQL to retrieve, maintain, and administer the database resources. Application code is written using Visual Basic for Applications (VBA) for the MS-Access solutions, and a combination of Visual Basic.NET and ADO.NET for the Oracle solutions. Modeling is done using either IBM Rational Rose or Microsoft Visio software.

The UML Data Modeling Profile (UML-DMP) (Gornik, 2003) is used to describe conceptual database models using stereotyped class diagrams. In UML-DMP, classes and associations play roles very similar to entities and relationships in ERDs. In addition, there are a number of concepts that are introduced in UML-DMP which are not available in classical ERDs, such as node, schema, database, and tablespaces.

Generally students who have taken the analysis and design class prior to the database course find the UML-DMP material easy to deal with and a good re-enforcement of concepts already learned. For those students lacking this background we have found that the compactness and simplicity of UML-DMP representations is manageable, and students are able to learn the data modeling strategy and tools very quickly. For example, Figure 2 shows a fragment of the library case introduced in the previous section. The figure shows the class diagram at the top, and its equivalent UML-DMP representation at the bottom. The logical version of the Borrower and the Book classes is transformed into a physical map showing three tables T_Borrower, T_Book, and the new file T_Loan. Observe the conversion of the many-to-many link into a pair of 1-to-many associations in the physical model, as well as the occurrence of stereotypes signaling primary key (PK), foreign key (FK), and Trigger constraints. Although we made minor changes to the table definitions, most of the work was done automatically by Rational Rose.

Clearly, UML-DMP modeling is at least as expressive as ER modeling, and we argue that the degree of difficulty associated with learning the tool is essentially the same as that of learning ERDs. Therefore, the database students who lack the recommended analysis and design background are not substantially disadvantaged with respect to their counterparts.

Our strategy when teaching database implementation is to minimize the number of UML charts used to describe a system. Under this minimalist pedagogical approach only a few types of representations are given to the students, namely use case, class, and sequence diagrams.

Use case diagrams are presented as a succinct depiction of the main functions of the system showing its services at a high and consistent level of resolution. Identifying the functions and the actors of a system is an essential part of the design effort, and fortunately, students – with or without previous UML experience – tend to see this as a relatively natural and workable step.

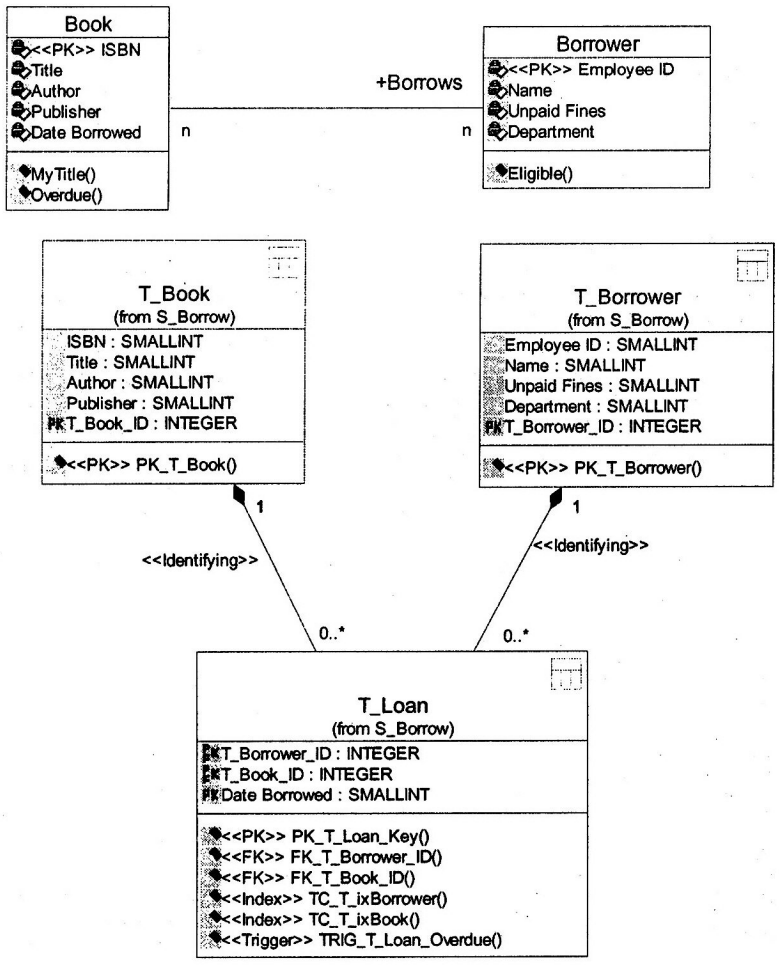

Figure 2. An UML-DMP fragment of the Library Case

Sequence diagrams have already been used in either the design or the advanced programming course. The sequence diagrams offer the students a dynamic description of what happens in a process at a high level of detail. As with use case diagrams, student understanding of these types of diagrams occurs very quickly and easily.

A major effort is made in learning how to represent the designer's perception of the system at hand using UML-DMP diagrams. A typical teaching strategy to deliver this topic is the following:

- In class, we present a simple and familiar example to which students can relate without many explanations, such as the library problem introduced earlier.
- After seeing a number of examples in class, students are given homework using a simple problem involving a small number of classes and associations, such as Purchase Order, Soccer Tournament, Library System, Bureau of Motor Vehicles, Video Rental, Employee Project, Employee Education Level, etc.
- Students are first asked to draw the UML-DMP data model by hand. After class discussion, a common correct solution is collectively agreed upon, and students use the computer software (IBM Rational Rose or MS-Visio) to represent the chosen solution (Figure 2).
- A representative portion of the system is selected. For example, in the case of the library problem, we may choose to describe the details telling how a person borrows a book. This description is pictorially represented as a sequence diagram (see figure 3). (Although practitioners generally do not show response messages to the extent shown in this example, we want to emphasize to the students where the system must wait for a response.)

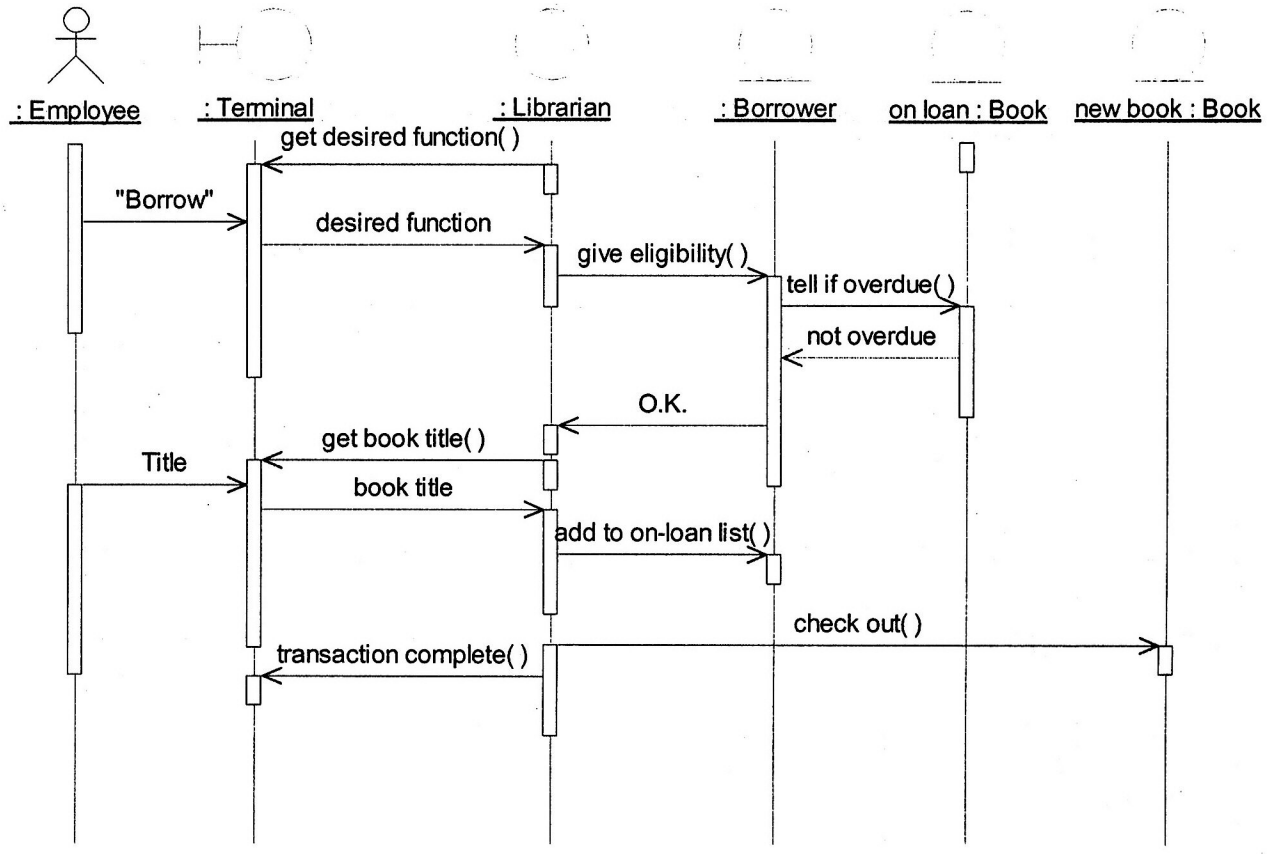

Figure 3 – Sequence Diagram for Borrowing A Book

- In the final phases of implementation, forms, reports, and code are produced to materialize the specifications of the system.

Today, most IS curricula use a relational solution such as Microsoft Access, SQL-Server, IBM DB2, Oracle, MySQL, Sybase. The UML-DMP components listed in Table 1 are mapped to the MS-Access and Oracle environments – used at CSU – as shown in Figure 4.

4. FUTURE PLANS

Figure 1 (Section 1) shows the current structure of the Information Systems curriculum. Except for the capstone course (IST 465), later courses are independent of the systems analysis course. This structure provides maximum flexibility for students and generally reflects the typical view of the system development process, in that programming, database design, network management, and systems analysis tend to be viewed as more or less independent activities once requirements have been defined and a basic logical model has been created.

However, increased faculty experience with object-oriented methodologies has led us to a more integrated view of the development process. That is, programming, database design, network implementation, Web site development, etc. are now viewed as specialized but related activities that should build on a single comprehensive analysis model. To reflect this development philosophy we intend to revise the curriculum as shown in figure 5.

Ideally, we would prefer to make systems analysis a prerequisite for the intermediate programming course as well, since this would allow us to include formal systems design techniques into the programming course. However, as a practical matter of curriculum planning, students need to take at least one course concurrently with systems analysis. The scheduling issues previously discussed are very real and we cannot have long prerequisite lists, but after several years experience with the current curriculum, it is our opinion that the advantages of requiring students to take systems analysis

UML-DMP Component	MS-Access GUI interface	ORACLE Implementation SQL command

Table	See Figure 4-a.	`CREATE TABLE T_Borrower (`
		`  EmployeeID  SMALLINT NOT NULL,`
		`  Name  SMALLINT NOT NULL,`
		`  UnpaidFines  SMALLINT NOT NULL,`
		`  Department  SMALLINT NOT NULL,`
		`  T_Borrower_ID  SMALLINT NOT NULL,`
		`  CONSTRAINT PK_T_Borrower PRIMARY KEY (T_Borrower_ID),`
		`  CONSTRAINT TC_T_Borrower CHECK (Department>99)`
		`);`

View	Named Query	`CREATE VIEW Late_Borrowers AS (SELECT ...FROM...WHERE...)`
Column	Fig. 4-b.	`Part of the Create Table command, for example` `  EmployeeID SMALLINT NOT NULL,`
Index	Fig. 4-c.	`CREATE INDEX IXBorrower ON T_Borrower ...`
<<PK>> Primary Key	Fig. 4-d.	`ALTER TABLE T_Borrower ADD CONSTRAINT PK_T_Borrower` `  PRIMARY KEY (T_Borrower_ID)`
<<FK>> Foreign Key	Fig. 4-e	`ALTER TABLE T_Loan ADD CONSTRAINT FK_T_Borrower` `  REFERENCES T_Borrower(T_Borrower_ID);`

<<Trigger>>	Not available	`CREATE OR REPLACE`
		`TRIGGER TRIG_T_Loan_Overdue`
		`BEFORE  INSERT OR UPDATE ON T_Loan`
		`BEGIN`
		`  --Code to check for fines and raise application error if needed`
		`END;`

Check Value Verification	Fig. 4-f	`CONSTRAINT TC_T_Borrower CHECK (Department>99)`

Table 1. UML-DMP components mapped on Microsoft Access and Oracle.

before database design outweigh the scheduling disadvantages.

Even with this compromise, the new program will allow us to present database design as a functional successor to the task of preparing a logical system model. With this view of system development, it necessarily follows that the database designer should begin his/her work from the logical model developed by the system analyst. Since the analyst develops the model in UML, it also follows that the database designer should be using UML, not ERDs.

5. CONCLUSION

We believe that the use of ERDs as the primary database design tool is being replaced by the use of UML diagrams.

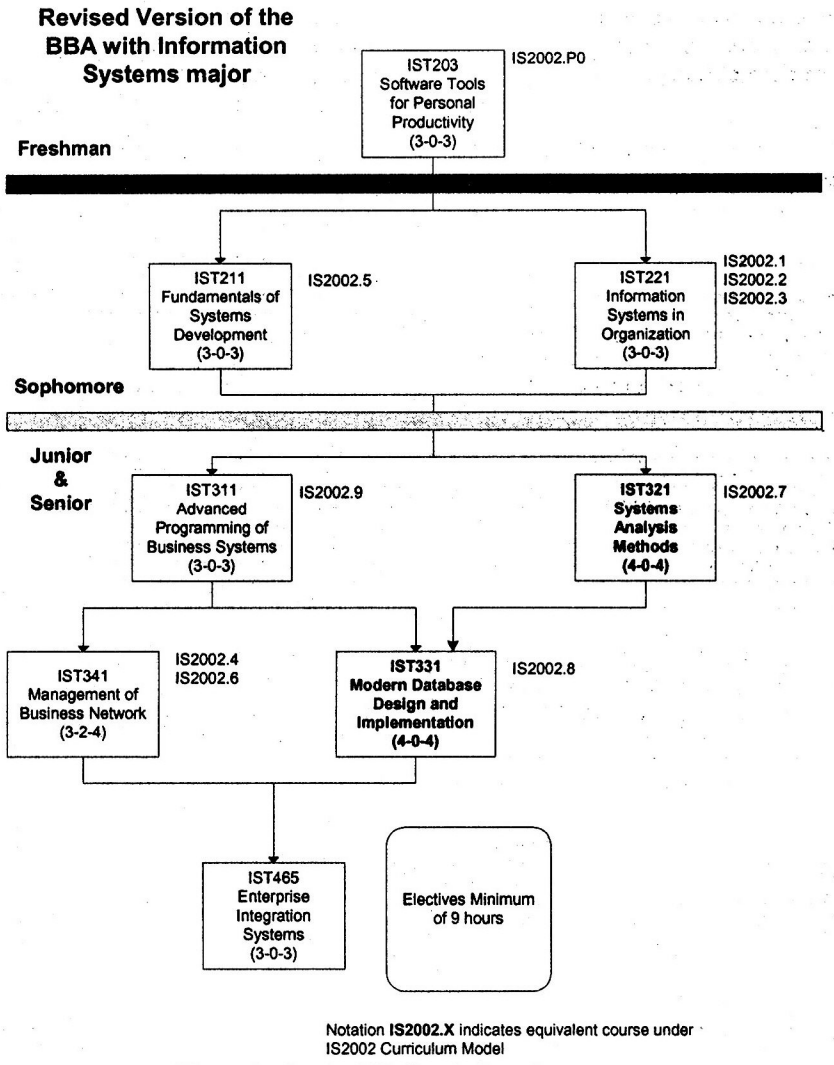

Figure 5 – Revised IS Curriculum Structure

Therefore, we have changed our database design course so that the primary tool is UML; although ERDs are still taught, their use has been reduced substantially. We must emphasize that this change is not simply a matter of style, or due to the increasing use of object-oriented software. Rather, it reflects the growing understanding that for new systems, database design is not an independent process but is an integral part of the overall system development process. UML reflects this view while ERDs do not. Furthermore, ERDs have an inherent weakness in that they do not support the specification of business rules and the resulting data behavior, while UML includes behavior specification as a fundamental component of the system model. As information systems and the databases that they require become increasingly complex, it becomes increasingly difficult to provide appropriate design specifications with ERDs. Therefore, it is vital that students learn how UML is used throughout the system development life cycle, including the design and implementation of databases.

However, in spite of the increased use of UML and the Data Modeling Profile, the use of ERDs is by no means a dead issue. For example, David Kroenke has virtually buried all references to UML diagrams in an appendix in the latest (10^{th}) edition of one of his text books (Kroenke, 2006). Also, ERDs have been used extensively to design and document existing databases, and are still used in many organizations to develop new databases or modify existing ones. An IS program would do a great disservice to its students by omitting all use of ERDs. None the less, it is our believe that, as object-oriented methodologies (particularly UML) are used more and more extensively to develop new systems, practitioners who become familiar with these methodologies will find little reason to use ERDs as well.

6. ENDNOTES

[1] © 2005, Computer Associates, Inc.
[2] © 1999, A component of PowerDesigner 7.0, Sybase, Inc.
[3] © 2002, Oracle, Inc.

[4] © 2004, Visible Systems Corporation
[5] IBM Rational Rose software is available to universities as part of the IBM Scholar program.
[6] "borrower" refers to a person, while "Borrower" is the class that represents borrowers.
[7] © 2005, InterSystems Corporation

7. REFERENCES

Chen, P. (1976) "The Entity Relationship Model – Toward a Unified View of Data," TODS, Vol. 1, No. 1.

Davis, G., Gorgone, J., Cougar, J.D., Feinstein, D., and Longenecker, H. (1997) "IS '97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information systems." Association of Information Technology Professionals.

Dorsey, P. (1998) "Introduction to Object Modeling in Oracle8," <www.dulcian.com/papers/Intro_to_Object_ → Modeling_in_Oracle8.htm>, p. 14.

Gornik, D. (2003) "UML Data Modeling Profile," IBM-Rational Software, Technical Article TP165, 05/02.

Gorgone, J., Davis, G., Valcich, J.S., Topi, H., Feinstein, D., and Longenecker, H., Jr. (2002) "IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information systems," Association for Computing Machinery, Association for Information Systems, Association for Information Technology Professionals.

Kroenke, D. (2006) Database Processing: Fundamentals, Design, and Implementation, 10th Edition, Prentice Hall, Englewood Cliffs, NJ.

Kroll, P. and Kruchten, P. (2003) The Rational Unified Process Made Easy, Addison Wesley, Boston, MA.

Page-Jones, M. (1988) The Practical Guide to Structured Systems Design, Yourdon Press, Englewood Cliffs, NJ.

Ramakrishnan, R. and Gehrke, J. (1998) Database Management Systems, McGraw-Hill, New York, NY.

Sharble, R.C. and Cohen, S.S. (1993) "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods," Software Engineering Notes, Vol. 18, No. 2, pp. 60-73.

Unified Modeling Language, Version 2.0 (2005), © The Object Management Group, Inc., <www.omg.org>.

Wilkinson, N. (1995) Using CRC Cards, SIGS Books, New York, NY.

Yourdon, E. (1989) Modern Structured Analysis, Yourdon Press, Englewood , NJ.

AUTHOR BIOGRAPHIES

Donald Golden is an Associate Professor of Computer and Information Science at Cleveland State University in Cleveland, Ohio. His primary research area is systems analysis and design, particularly with object-oriented systems.

Victor Matos is an Associate Professor of Computer and Information Science at Cleveland State University in Cleveland, Ohio. His research area is primarily database systems

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096