Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2005 Proceedings

Americas Conference on Information Systems (AMCIS)

2005

Practical Complexity in Adapting Object Oriented Approach of Systems Analysis and Design

Mohammad A. Rob
University of Houston - Clear Lake, rob@cl.uh.edu

Practical Complexity in Adapting Object-Oriented Approach of Systems Analysis and Design

Mohammad A. Rob
School of Business
University of Houston-Clear Lake
rob@cl.uh.edu

ABSTRACT

Recently, there has been a surge of interest in adapting object-oriented (OO) concepts, UML, and Unified Process of system development in the Systems Analysis and Design texts. However, there is a question of how to best fit these concepts with the existing coherent discussion of structured approach. This paper addresses some of the intricacies of OO concepts such as complexity of diagrams and models, weak links between phases, and lack of support for designing system components. We would like to recommend that there should be a separate text for the OO methodology and it should not present various OO models according to the phases of the traditional structured approach, rather it should focus on the evolution of the models leading to the design of system components. Furthermore, there should be a standard set of models for the OO methodology as well as a clear definition of steps as an analyst moves from one set of models to the next.

Keywords

Systems development life cycle, SDLC, systems analysis and design, structured development, object-oriented design, OO analysis, complexity.

INTRODUCTION

Systems Development Life Cycle (SDLC) outlines the necessary processes that include some phases and activities to successfully develop an information system. Almost all texts used for the Systems Analysis and Design (SAD) course typically discuss and elaborate a structured approach of systems development commonly known as Waterfall model (Dennis and Wixom, 2000, Hoffer, et al., 2000; Kendall and Kendall, 2001; Shelly, et al., 2003; Satzinger, et al., 2004; Whitten, et al., 2003). These texts also organize their chapters sequentially according to the phases (planning-analysis-design-implementation) and activities of the SDLC as followed in the Waterfall model. In recent years, there has been a rush of including Object-Oriented (OO) approach of systems development in the SAD texts. Some texts adopt OO models and techniques in various chapters in a comparative manner along with the structured approach (Satzinger, et al., 2004; Whitten, et al., 2003), while others include them at the end of the texts as an introduction to a new methodology (Dennis and Wixom, 2000; Shelly, et al., 2003). Some authors even came up with an OO version of their SAD texts (Dennis et al., 2002; George, et al., 2004). There is a dilemma of how to best fit the object-oriented topics with the existing coherent discussion of structured approach. The question is whether the OO methodology should be included along with the traditional approach in the same text or they should be addressed separately. This paper tries to find some answers to this question by addressing some of the intricacies of OO methodology that are prohibiting authors as well as practitioners to adopt this methodology as the primary approach of systems analysis and design. First, we will provide a brief overview of the OO concepts and the Unified Process of systems development as compared to the traditional approach. Then we put forward several reasoning for not accepting OO methodology as the latest and greatest method of systems development. Next we provide a brief perspective to the reader regarding the approaches of teaching systems analysis and design in the Software Engineering program as compared to the MIS program. Finally, we conclude with some specific recommendations.

RESEARCH BACKGROUND

In a recent study, conducted in three consecutive semesters (Fall, 2002 - Fall, 2003), graduate students in a traditional SAD course were asked to write a paper on a topic related to the activities of the systems analysis and design but that are not discussed in detail in the course. The majority of the students wrote on topics related to the object-oriented approach of

systems analysis and design. According to studies by Hardgrave and Douglas (1998), more and more Information Systems departments are teaching OO topics in their curricula. Employers are also looking for analysts with knowledge on object-oriented analysis and design (Hotjobs, 2004; Monster, 2004). Thus object-oriented methodology cannot be ignored in teaching Systems Analysis and Design.

TRADITIONAL VS. OBJECT-ORIENTED APPROACH TO SYSTEMS DEVELOPMENT

A systems development project goes through a sequence of some fundamental phases such as planning, analysis, design, and implementation. Each of these phases can be divided into a series of steps or activities that rely on some models and techniques to produce required deliverables. Even though all projects cycle through some common phases or activities, but how they are approached by the systems development group can be different – the project team might move through the phases or steps logically, consecutively, incrementally, or iteratively (Dennis, et al., 2002; Satzinger, et al., 2004).

The Traditional Approach

The most common approach to system development is a structured methodology based on Waterfall model. It adopts a formal step-by-step approach to the SDLC phases and activities – the activities of one phase must be completed before moving to the next phase. The structured approach looks at a system from a top-down view. At the center of this approach is the process model, which depicts the business processes of a system, and the primary model that presents the processes is the data-flow diagram (DFD). Various levels of DFD are developed to understand the details of business processes. The DFDs and their associated data dictionary contain information about the systems components (inputs, outputs, processes, and data storage) that need to be designed and ultimately built. The structured approach has been around for many years and almost all SAD texts describe this model in detail – the sections and chapters of these texts logically follow the phases and activities of SDLC (Dennis and Wixom, 2000, Hoffer, et al., 2000; Kendall and Kendall, 2001; Satzinger, et al., 2004; Whitten, et al., 2003; Shelly et al., 2003).

The Object-Oriented Approach

Another approach to systems development that is widely discussed in recent years is the object-oriented (OO) methodology (Booch, et al., 1999; Brown, 2002; Dennis et al., 2002; Schach, 2004). It is developed by the software engineering professionals who deal with large and complex systems in domains such as aerospace and process control (Schach, 2004). Business system is only one domain that is typically addressed in the SAD texts.

The object-oriented approach views a system as a bottom-up approach to systems development. To start with, it describes the system through a set of business processes it performs as well the object classes that these processes deal with. It uses a set of diagrams or models to represent various views and functionality of a system which are commonly known as the Unified Modeling Language or UML. When these models are used along with a particular method of systems development, the OO approach became known as the Unified Process (Schach, 2004). Unified Process follows an iterative and incremental approach to systems development. The systems development life cycle is viewed as consisting of several increments or phases: inception, elaboration, construction, and transition (Booch, et al., 1999; Dennis, et al., 2002; Schach, 2004). In each increment or phase, the developers move through the activities of gathering requirements, analyzing the requirements, designing the system, implementing the design, and testing the system. Thus the Unified approach is a two dimensional model as compared to the one-dimensional traditional waterfall model. See Figure 1. As shown, the phases of the traditional systems development approach do not match with those of the Unified Process; but in each increment, all phases of the SDLC (requirements, analysis, design, implementation, and testing) are visited until the developers satisfy the requirements. However, in each increment, activities of one phase predominate over the others – causing the systems development effort to move from the inception to elaboration, from elaboration to construction, and from construction to transition.

It is important to note that many authors use the terms object-oriented approach and UML interchangeably, as UML covers universally accepted OO models, notations, and steps to develop an information system. UML focuses on three architectural views of a system: functional, static, and dynamic. The functional view describes the external behavior of the system from the perspective of the user. Use cases and use-case diagrams are used to depict the functional view. The static view is described in terms of attributes, methods, classes, relationships, and messages. Class-responsibility-collaboration (CRC) cards, class diagrams, and object diagrams are used to portray the static view. The dynamic view is represented by sequence diagrams, collaboration diagrams, and statecharts. All diagrams are refined iteratively until the requirements of a proposed information system are fully understood, and the design is laid out. Finally, the information system is developed through a combination of traditional relational database and object-oriented programming language. Due to factors such as a large number of models, the relationships of these models in various viewpoints of the systems architecture, a new approach to

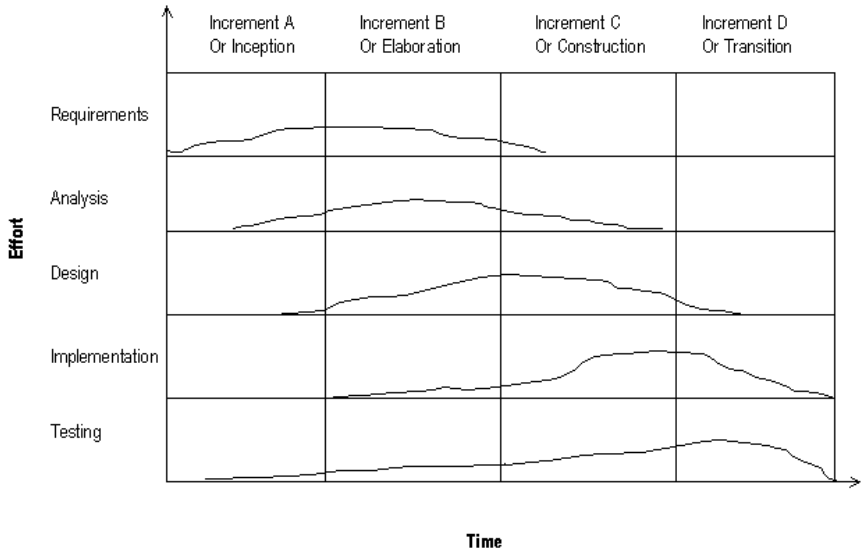

Figure 1. Iterative and Incremental Model of the Unified Process

system development, and a large vocabulary of UML, most authors present only a subset of UML knowledge in their texts, and it is presented as the object-oriented methodology.

DILEMMA BETWEEN THE TWO METHODOLOGIES

There is a common misunderstanding that the object-oriented approach is the latest methodology of systems analysis and design. The OO methodology promises many benefits, but excessive hype has lead to unrealistic expectations among executives and managers (Shah, et al., 2004). Even software developers often miss the subtle but profound differences between the OO approach and the classic systems development approach. Thus, it is not clear whether we all should move towards the OO methodology of systems development or not. According to a report by Sircar et al. (2001), IT managers revealed that 39% of organizations have adopted OO methodology in some form; nevertheless, only 5% of IT projects are developed in OO methodologies. According to Schach (2004) the Unified Processes are intended for use in developing large and complex information systems; however, handling many intricacies of the UML might be far more complex than the development of the most information systems in business.

There have been some studies on the use of OO methodology as compared to structured approach; although most of these were performed in the classroom setting. Johnson (2002) provides an excellent overview of 12 empirical studies that report advantages and disadvantages of using OO methodology as compared to structured methodology. These studies include subjects from both classroom setting as well as IT professional arena. Most studies show that OO methodology is difficult to learn than conventional methodology; however, it produces increased quality and productivity. Sim and Wright (2002) performed a study on a group of students regarding their understanding of basic OO concepts in three categories, namely the structural, behavioral, and modeling. They concluded that the students can grasp the structural concepts such as objects and classes easily as compared to dynamic aspects of objects such as operations, message passing, object models, object interaction diagrams, and so on.

It is also important to mention that most of the studies mentioned above were performed only on the learning difficulty of OO concepts and models used for the analysis and design activities – not for the complete life cycle activities, such as the designing and implementing the system components. These studies were also limited to a set of concepts and models – not all the concepts and models used in the OO paradigm were addressed. Most authors suggest that more research is necessary on the subject, especially in industrial settings with real-life projects rather than semester-long student projects. The following discussions focus on some of the intricacies of adapting the OO methodology as compared to the structured approach in a systems development project.

Complexity of Diagrams and Models

There has been some discussion of UML complexity in the literature. Grossman et al. (2004) performed a web-based survey to understand how UML users perceive the overall technology to fit with their understanding and knowledge. They concluded that there is still much to be learned about UML to perceive its benefit and users still do not have enough of a feeling of how this technology fits with the tasks they are trying to perform. Erickson and Siau (2004) tried to explain the user perception of UML complexity by considering individual OO diagrams and their associated constructs (such as classes, objects, methods, roles, etc.) that constitute a diagram. Through a measure of relative importance for nine OO models, they have concluded that Object Diagram is the most complex and Class Diagram is the least complex. The survey results also conclude that users consider four OO diagrams (Class, Use Case, Sequence, and Statechart) as most important, indicating that practitioners use OO methodology mainly for the analysis and design stages of a system development effort and not for the coding and testing stages. In this paper, we try to present the UML complexity by considering the relationship between the various OO models used in various stages of systems development, and we present this in comparison to that of the models used in the structured approach.

There are mainly three diagrams or models used in the structured methodology: a data- flow diagram, an entity-relationship diagram, and a structure chart. The relationships between these models are very simple; and the latter two diagrams are developed from the data-flow diagram. For example, the entity-relationship diagram is developed from the data stores used in the data-flow diagram, while the program modules and data-flows are used to develop the structure chart. In the OO methodology, there are about thirteen diagrams and models to deal with, namely: essential use case description, use case scenarios, use case diagram, collaboration diagram, CRC cards, classes, class diagram, object diagram, sequence diagram, CRUD matrix, state chart diagram, package diagram, use scenarios, real use cases, and windows navigation diagrams (Dennis et al., 2002). Whitten et al. (2003) consider nine diagrams including activity diagrams, component diagrams, and deployment diagrams. Understanding the evolution of these models and their relationships with each other is important for the successful design and development of an information system.

In Figure 2, we have organized these models in a workflow diagram to demonstrate the complexity of the evolution of various diagrams in the OO analysis and design. The gray smoothed rectangles represent the essential diagrams or models drawn in the analysis phase and the black smoothed rectangles represent those in the design phase. It is important to note how one model is developed from several models and descriptions. Dennis et al. (2002) contend in their object-oriented SAD text that because of the complexity of the Unified and Open approaches, they have followed a minimalist style of presenting a generic approach of OO analysis and design. On top of the complexity of the models, one has to consider the iterative nature of the OO methodology – that is, each of these models and descriptions change as more and more requirements are gathered. Each different style of model needs a different style of thinking for someone to create one or to work with it (Brown, 2002).

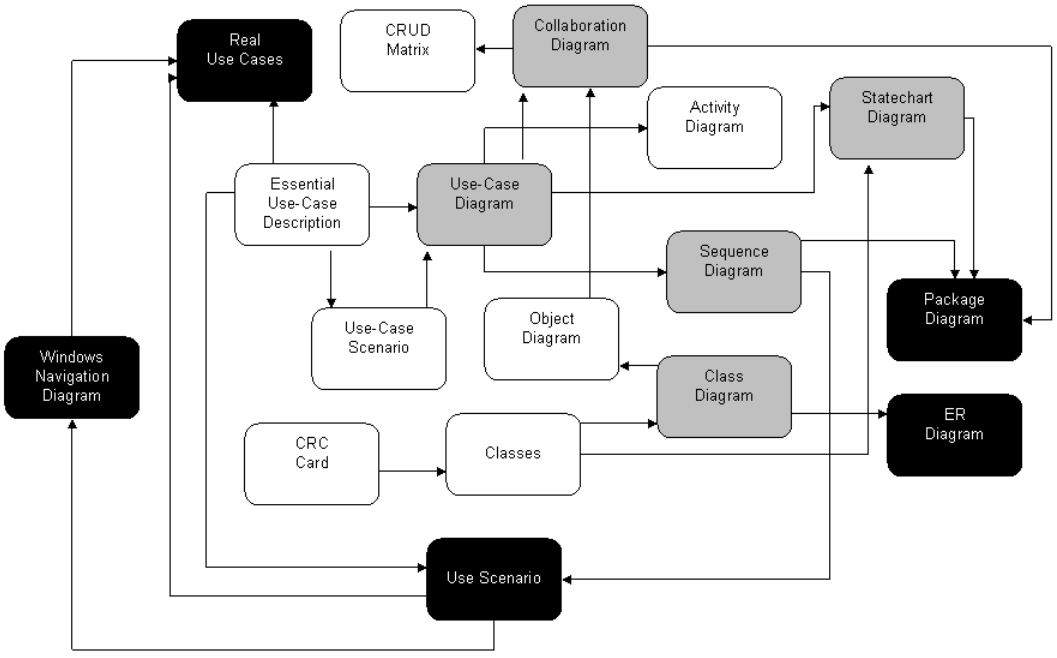

Figure 2. Relationship between the various models of the OO approach

Relationship between the Phases

While the structured approach provides clear-cut steps in moving between the phases from the beginning to the end of the systems development life cycle, the OO approach does not; or the steps are so complex and time-consuming that most authors fail to clarify them. The iterative and incremental nature of object-oriented approach refers to continuous testing and refinement of the system throughout the project lifecycle. The same UML diagramming techniques are used throughout all phases of the SDLC, although some diagrams are more important in some phases than the others. Some authors discuss a model in the analysis phase, while others discuss it in the design phase. For example, Dennis et al. (2002) discuss about collaboration diagram and statechart diagram in the analysis phase while they are in the design phase according to Satginger et al. (2004). According to Dennis et al., (2002), the system design models in the OO perspective simply refine the system analysis models by adding system environment or solution domain details to them and refining the problem domain information already contained in the analysis models. As the developers move through the SDLC activities, the diagrams gradually become more detailed, blurring the separation between the analysis and design phases. Understanding the continuous analysis of a problem through Use Cases is fine, but when it comes to design, it becomes very complex – many models to deal with (Schach, 2004). What we need is a clear understanding of how the models evolve; rather than dividing them into analysis and design phases.

Most authors who present OO methodology in SAD texts spend considerable amount of time discussing various models to analyze user requirements, but often present poor discussion of design phase or provide poor correlation between the models used in the analysis phase and those used in the design phase. Dennis et al. (2002) mention that 'packages' are based on use case diagram, class diagram, sequence diagram, collaboration diagram, activity diagram, and so on, but they do not provide any step-by-step process of evolving the packages from these diagrams – making OO methodology an overwhelming effort to understand not only by the analysts but also by the authors. Brown (2002) just mentions that a package or subsystem is a group of classes, subsystems and the like, that have a well-defined small interface to the rest of the system and are treated as a unit. Understanding the requirements of a system through OO approach seems to be clear and logical, but when it comes to designing a system, the OO methodology seems to be confusing. Satizinger and Jackson (2003), who introduced OO concepts in their SAD text (Satizinger et al., 2004), contend that fundamental OO concepts and techniques are addressed in systems development texts; however, a glaring weakness is the lack of useful guidelines and strategies of taking the high level OO requirements models into implementable architecture and detailed design.

Designing System Components

The ultimate goal of any systems development approach is to effectively design the system components such as inputs, outputs, program modules, and database, so that programmers can develop those components. Figure 3 illustrates how the design of the system components evolves from the DFDs in the structured approach. Once the data flow diagrams are completed, a data dictionary is developed; and based on the DFDs and data dictionary, design of various system components evolves logically. This is illustrated in Figure 4 through the dissection of a data-flow diagram. As shown, data stores are moved out of the DFD, and the data storage is designed using the entity-relation diagram. Similarly, the DFD processes and internal data flows are used to design program modules using structure charts. The input/output screens and reports are developed from the definitions of input/output data flows.

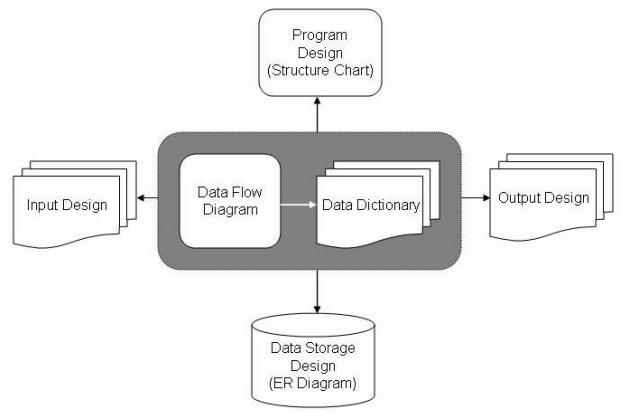

Figure 3. Evolution of the design of system components in the structured approach

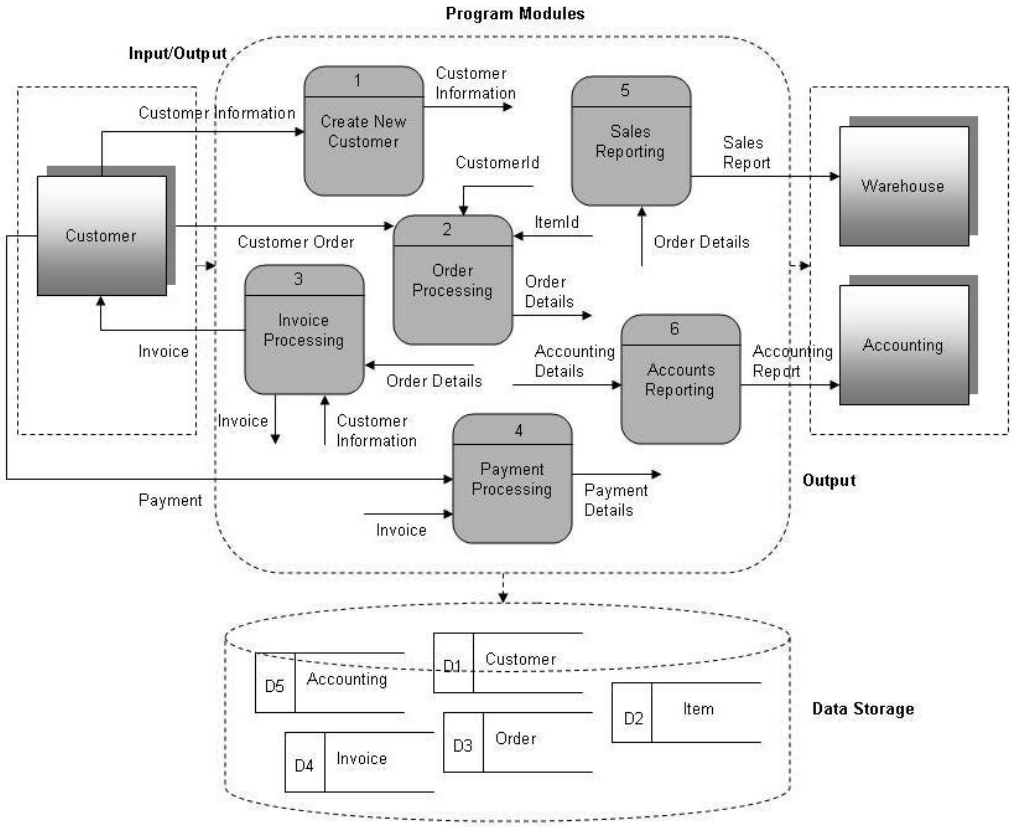

Figure 4. Evolution of the design of system components from a Data Flow Diagram in the case of the structured approach

It is not easy to draw diagrams like Figures 3 and 4 for the OO design. The only information that is easily extractable is the data storage definition from the class diagram; although a class diagram contributes to both program design and data storage design. The input/output information is extracted from many sequence diagrams and class definitions (Satzinger, et al., 2004). There is no equivalent of structure chart for program design in the OO methodology. The closest equivalent is the package diagram, which is developed from class diagrams, sequence diagrams, statechart diagrams, and collaboration diagrams. There is no single repository like data dictionary in case of structured approach, which holds all data for input, output, data storage, and process description. However, the three-layer approach of a system during the design phase containing a view layer, a domain layer, and a data access layer is more appropriate for today's business system as well as the development environment.

We have tried to develop a diagram like Figure 3 in case of the OO methodology that depicts the steps required in designing the system components. This is illustrated in Figure 5, which is developed from Figure 2; however, emphasis is placed on the models especially the design models. Without getting into the details of any model, one can just understand the complexity of designing program and user-interface components by looking at the number of arrows originating from various models and leading to these components.

Pitfalls in the Implementation

Moving from the design to the implementation phase in the OO approach further complicates the matter. Why should we talk about the class model at the beginning of the analysis phase and then discuss about the relational database in the implementation phase; why not implement the system in an object-oriented database? According to Satginger et al. (2002), the object-oriented approach views an information system as a collection of interacting objects that work together to accomplish tasks - there are no process or programs; there are no data entities or files. This statement seems to be contradicting when authors discuss about the relational database in the design phase and continues that in the implementation phase. Use cases and sequence diagrams are very useful in understanding events triggered by external agents through user

interfaces. They provide an understanding of the user interface as well as the program codes of a system – leading to the implementation of these components using already known object-oriented languages.

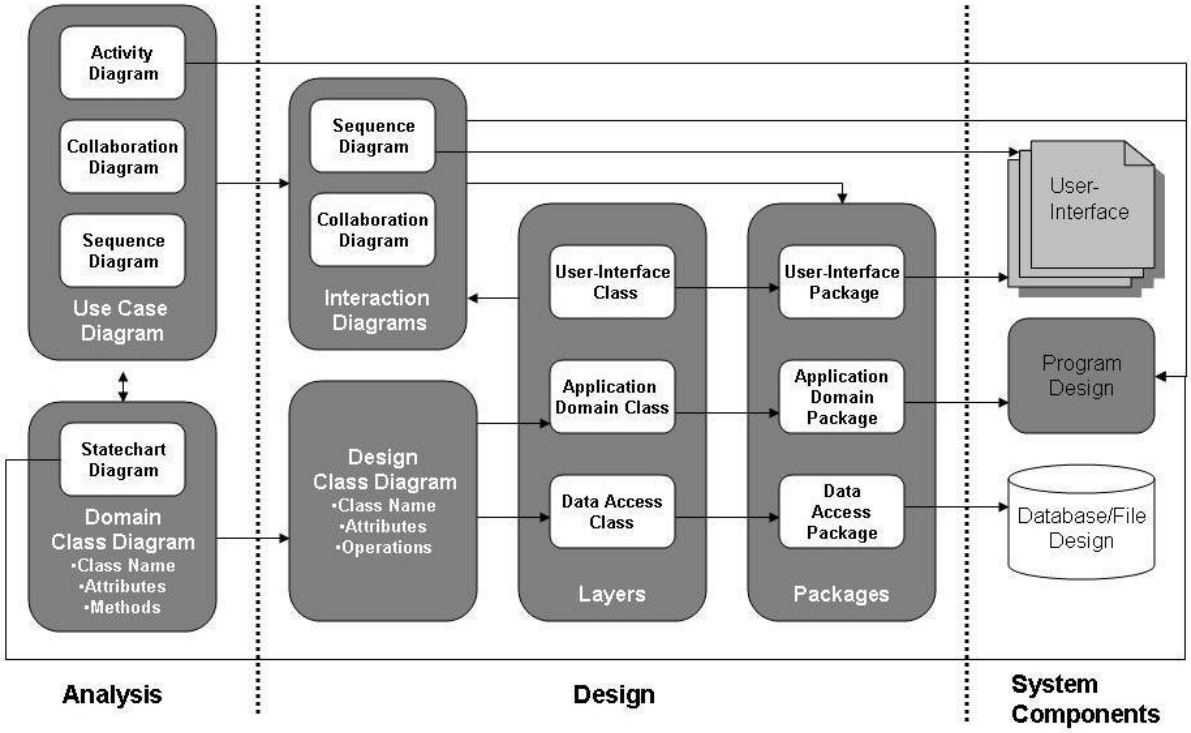

Figure 5. Evolution of systems design components from the analysis and design models in case of the OO approach

The depiction of data for a business system through a class diagram sometimes seems to be more of educational than realistic. In the object-oriented text, Dennis et al. (2002, p.195) wrote, "During analysis, classes refer to the people, places, events, and things, about which the system will capture information. Later, during design and implementation, classes can refer to implementation-specific artifacts like windows, forms, and other objects used to build the system." The first sentence refers to data typically in a database and second sentence refers to user interface. Thus, the statements provide confusion as to whether the class diagram is for the user interface, or the database, or both. Development of the design classes (user interface, application domain, and database) from the analysis classes is not clear in many texts. The class diagram also contains methods an individual object can perform; however, it is only applicable to the application domain. Many authors fail to clarify this fact. The object diagram depicts interaction between individual objects, which is only applicable to the application class, not in the data class if data are stored in rows and columns.

Furthermore, in the OO methodology, new classes are added to the business or domain-related class diagram for user-interface, other system networks, and communication links. However, most authors only elaborate the user-interface class, which is well understood by programmers, as it is part of the development environment. Defining buttons, text boxes, radio buttons, as well as events triggered by these controls is a knowledge gathered in a programming course – repeating this discussion in a SAD text not only increases the volume but also looses sight of the business domain to be modeled. Furthermore, the OO approach goes into considerable detail of how a program should be developed. It is important to have a description of the business process as well as the associated programs, but how it should be implemented can be left to the programmers.

Comparison between the two Approaches

According to Dennis et al. (2002), UML is nothing more than a notation – it does not dictate any specific approach to developing information systems. Even though it is unlikely, it is possible to develop an information system using a traditional approach, such as structured systems development or information engineering, and to document the analysis and design using

the UML. This reflects the intricacies of using true OO methodology in developing an information system. Table 1 summarizes some advantages and disadvantages of traditional and OO approaches to systems development.

Structured Approach	Object-Oriented Approach
Logical steps of SDLC: Analysis, Design, and Implementation	Repeating all phases of SDLC during each iteration
Few models to deal with	Many models to deal with
There are clear-cut documentation at the end of each phase	There are no documents per see; all information is contained within the model descriptions
Focuses mainly on the business aspects of a system and deals with other components such as user interface, network architecture, processing architecture separately	Starts with the business aspects of a system but deals with other components such as user interface, network architecture, processing architecture together as the analyst moves from the requirements models to design models
A small and consistent vocabulary to follow through the life cycle	A large and changing vocabulary during various phases of the life cycle
Process models are the focus of understanding user requirements	Use cases are the focus of understanding user requirements
Easy to from analysis models to design models	Design models depend on many analysis models
Easy to extract design information for systems components such as user-interface, application programs, and database or files	Very complex to extract design information for systems components
Business logic of a process is described by the process description using structured English, decision tables, and decision trees	Business logic of a process is described by use case scenario, use case description, interaction diagram, and activity diagram
Does not dictate any model for the development environment	Three-layer approach to systems design is closely analogous to a development environment
Identification of input and output to the system is simple and they are extracted from the input/output data-flows in the data-flow diagram	Input and output information are scattered in many sequence diagrams as input/output messages. Separate classes describe the data
All data necessary for designing and developing a system is found in a single repository called data dictionary	There is no single repository for data; they are scattered with class definitions and use case descriptions
Development of programs through subroutines and functions are left to the programmers at the implementation phase	Development of programs is conceived at the requirements phase through defining classes, methods, and messages, and continues through the design phase
Detailed programming knowledge is not necessary for successful analysis and design	Detailed programming knowledge is necessary for successful analysis and design
Appropriate for developing documents and then start programming; hence programming can be outsourced	UML iterative approach requires continuous development and testing; hence programming can not be outsourced
Easy to manage systems development project, as tasks are defined in phases, through output documents, and especially the hardest part - programming, which is defined through program modules in a top-down fashion	Not easy to manage systems development project as models require continuous revisiting, and the hardest part which is programming that depends on packages - are complex; thus task duration is hard to quantify

Table 1. Comparison Between the two Approaches of Systems Development

DILEMMA BETWEEN THE SAD AND SOFTWARE ENGINEERING TEXTS

Software Engineering (or *Software Design*) is a text typically used in an introductory course in the Software Engineering program (Budgen, 2003; Pressman, 2000; Sommerville, 2000). The topics covered in this course are subsequently elaborated in courses such as requirement engineering, software engineering processes, and verification and validation. Thus there are separate texts for each of these courses. This defines the vigorousness necessary for the software engineering processes in operation- or mission-critical systems. Typically, there is only one Systems Analysis and Design course taught in an MIS program and also recommended by some MIS curricular models (Davis, et al., 2002; Longenecker, et al., 1999). Only the *Software Engineering* text can be compared with the Systems Analysis and Design text. According to Schach (2004), "the Unified Process is intended for use in developing large, complex information systems. In order to be able to handle the many intricacies of such information systems, the Unified Process is itself large. It would be hard for an instructor to cover every aspect of the Unified process in a single course." Incorporating OO approach within the traditional approach in the same text further complicates the subject matter and confuses the reader.

CONCLUSION

We have discussed some of the intricacies of the object-oriented concepts and the Unified Process of systems development methodology as compared to the structured approach. In the structured approach, there are only three models to deal with, while in the OO approach there are about thirteen models, and these models evolve as an analyst moves from analysis to design. UML defines a large set of diagramming techniques, but most SAD texts focus on a smaller set of most commonly used techniques due to its complexity. In the era of object-oriented programming, object-oriented modeling is desirable, but what we are lacking is a clear definition of steps as an analyst moves from the activities of analysis to design and from design to implementation. There are so many diagrams for each business function, that in all probability the analysts and programmers will so get bogged down with diagramming, that they will fail to see the implementation of all the diagrams.

In the structured approach, design of system components such as input, output, program modules, files and database evolve logically from the DFDs, while in the OO approach there are many models to deal with and most authors fail to clarify the steps of designing the systems components. As the ultimate goal of a systems analysis and design methodology is to design the system components, there should be clear definitions of how to extract required information from the various OO models that can lead to the designing of the system components.

Furthermore, the structural methodology and Unified Process are two completely different paradigms of systems development methodology. The phases of the traditional approach do not match with those of the Unified process. Thus overwhelming a student with two different approaches and with a large number of models may not serve the purpose of acquiring the full concept of OO methodology; rather it might have an adverse effect with a conclusion that OO methodology is complex and convoluted. Furthermore, mixing up the structured methodology and the OO methodology in a single text not only confuses the students but the authors as well.

In conclusion, we would like to recommend that if we are to teach the object-oriented approach, we should have a separate text for it and it should not present various OO models according to the phases of the traditional structured approach, rather it should focus on the evolution of the models leading to the design of system components. Furthermore, there should be a standard set of models for the OO methodology as well as a clear definition of steps as an analyst moves from one set of models to the next.

REFERENCES

1. Booch, G, Jacobson, I. and Rumbaugh, J. (1999), *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA.
2. Brown, D. (2002), *An Introduction to Object-Oriented Analysis: Objects and UML in Plain English*, John Wiley & sons, New York.
3. Budgen, D, (2003), *Software Design, Second Edition*, Addison Wesley, New York.
4. Davis, G.B., Feinstein, D., Gorgone, J.T., Longenecker, Jr., H. E., & Valacich, J.S. (2002), IS 2002: An Update of the Information Systems Model Curriculum, *Proceedings of the Sixteenth Annual Conference of the International Academy for Information Management*, New Orleans, LA, pp. 76-82.
5. Dennis, A. and Wixom, B. H. (2000), *Systems Analysis and Design*, John Wiley & sons, New York.

6. Dennis, A., Wixom, B. H. and Tegarden, D. (2002), *Systems Analysis and Design: An Object-Oriented Approach*, John Wiley & sons, New York.
7. Erickson, J. and Siau, K. (2004), "Theoretical and Practical Complexity of Unified Modeling Language: Delphi Study and metrics Analysis," Proceedings of the Twenty-Fifth International Conference on Information Systems.
8. George, J. F., Dinesh, B., Valacich, J. S. and Hoffer, J. A. (2004), *Object-Oriented System Analysis and Design*, Prentice Hall, Upper Saddle River, NJ.
9. Grossman, M., McCarthy, R. V. and Aronson, J. E. (2004), "The Unified Modeling Language: An inquiry into current practices and user perceptions," Proceedings of the Americas Conference on Information Systems, New York.
10. Hardgrave and Douglas (1998), "Trends in Information Systems Curricula: Object-Oriented Topics," Americas Conference on Information Systems.
11. Hoffer, J. A., George, J. F. and Valacich, J. S. (2000). *Modern Systems Analysis & Design*. Prentice Hall, Upper Saddle River, NJ.
12. Hotjobs (2004), Yahoo! Hotjobs, http://hotjobs.yahoo.com/, accessed on July 15, 2004.
13. Johnson, R. A. (2002) "Object-Oriented Analysis and Design – What Does the Research Say?" *Journal of Computer Information Systems*, pp. 11–15
14. Kendall, K. and Kendall, J. (2001). *Systems Analysis and Design, 5th Edition*. Prentice Hall, Upper Saddle River, NJ.
15. Longenecker, Jr., H. E., Feinstein, D. L., Haigood, B., and Landry, J. P. (1999), On Updating the IS.97 Model Curriculum for Undergraduate Programs of Information Systems, *Journal of Information Systems Education*, (10:2), pp. 5-7.
16. Monster (2004), Monster Job Search, http://www.monster.com/, Accessed on June 15, 2004.
17. Pressman, R (2000), *Software Engineering: A Practitioner's Approach*, Fifth Edition, McGraw Hill, New York.
18. Satzinger, J. W. and Jackson, R. B. (2003), Making the Transition from OO Analysis to OO Design with the Unified Process, *Communications of the Association for Information Systems*, Volume 12, pp. 659-683.
19. Satzinger, J. W., Jackson, R. B., and Burd, S. D. (2004), *Systems Analysis and Design in a Changing World, Third Edition*, Course Technology, Boston, Massachusetts.
20. Schach, S. R. (2004), *Introduction to Object-Oriented Analysis and Design with UML and the Unified Process*, Irwin- McGraw Hill, New York.
21. Shah, V., Sivitanides, M and, Martin, R. (2004), Pitfalls of Object-Oriented Development, http://www.westga.edu/~bquest/1997/object.html, Accessed on June 30, 2004.
22. Shelly, G. B, Cashman, T. J. and Rosenblatt, H. J. (2003), *Systems Analysis and Design, Fifth Edition*, Course Technology, Boston, Massachusetts.
23. Sircar, S., Nerur, S.P., and Mahapatra, R. (2001), Revolution or Evolution? A Comparison of Object-Oriented and Structured Systems Development Methods, *MIS Quarterly*, Vol. 25, No. 4, pp. 457-471.
24. Sommerville, I. (2000), *Software Engineering, 6th Edition*, Addison Wesley, New York.
25. Sim, E. R. and Wright, E. (2002), The Difficulties of Learning Object-Oriented Analysis and Design: An Exploratory Study, *Journal of Computer Information Systems*, Winter 2001-2002, pp. 95-99.
26. Whitten, J. L., Bentley, L. D. and Dittman, K. C. (2003), *Systems Analysis and Design Methods, Sixth Edition*, McGraw Hill, New York.