COMPONENT SOFTWARE: A NEW SOFTWARE ENGINEERING COURSE

H. Conrad Cunningham, Yi Liu, Pallavi Tadepalli, and Mingxian Fu
Department of Computer and Information Science
University of Mississippi
University, MS 38677 USA
{hcc,liuyi,pallavi,mfu}@cs.olemiss.edu

ABSTRACT

In recent years component-based software development has emerged as an important new set of methods and technologies. This paper describes a new course—Component Software—that focuses on component-based design using technology-independent methods and component implementation using Enterprise JavaBeans. The paper outlines the approach and structure of the course. It also presents a case study used in the course.

1. INTRODUCTION

In the early Twenty-First Century most software systems exist in highly dynamic environments. Their requirements change frequently and they must be built or modified on challenging development schedules. The software systems are large and decentralized. They execute in a distributed fashion and the development and maintenance of the software and data involved may be distributed among many groups in the enterprise. In this context, software systems need to be modular and easy to change; the development methods and technologies should support reuse of analysis, design, and tested code. The software development community is approaching the development of such distributed, enterprise-level software applications with component-based methods and technologies. The idea is that a new software application can be built quickly and reliably by assembling preexisting components with a few new components. Furthermore, it should be possible to handle changes in the requirements by replacing a small number of components.

To construct component-based software systems, software developers need a range of knowledge and skills. First, they need to know how to analyze problems so that they can identify, specify, and design effective components and component architectures. Second, they need a firm grasp of appropriate programming concepts and technologies, in particular use of object-oriented languages. Third, they need a conceptual understanding of and a facility in use of appropriate component technologies. As Clemens Szyperski has said, "Component Software Engineering is an emerging technology about to take the software industry by storm" [9]. As computing science educators, we need to help prepare our students for this storm by enabling them to use and build software components and component-based applications.

This paper describes a Component Software course that addresses, in part, the first and third of the above knowledge and skill needs. This course aims to present the concepts and techniques for design and implementation of component-based software, with a focus on technology-independent concepts and methods. The course assumes

that the student has a solid background in object-oriented programming concepts such as data abstraction, inheritance, composition, and polymorphism. In particular, the course assumes that the student has a good working knowledge of the Java programming language. The course uses a development approach similar to the UML Components methodology [2] and uses the Enterprise JavaBeans (EJB) component model and other Java 2 Enterprise Edition (J2EE) technologies [3] for examples and practical exercises. The course thus emphasizes large-grained, distributed components typically executing in a client-server architecture.

2. COURSE PRINCIPLES

The first guiding principle in the construction of this course is the *separation of concerns*. There are at least two aspects of this principle that are applied here.

The first aspect is the separation *of the product from the process*. By this we mean the separation of the *specification*, which describes "what" a component is required to do, from the *implementation*, which describes "how" the specification is realized (i.e., the algorithms, data structures, and program structures needed). The methods taught are independent of the details of the implementing technologies. The approach develops a sequence of system specification models and then maps the design specification into a particular technology. The implementation details of a component are hidden behind an interface that has been systematically developed from the requirements.

The second aspect of the principle is the *separation of the logic into architectural layers* with well-defined purposes. For example, we separate the presentation logic (i.e. the user interface issues) from the business logic (i.e. the application-specific data manipulations and processing). Carrying this further, we divide presentation logic into layers for the management of the user interface elements and for the user's dialog with the system within an interactive session. We also divide the business logic into layers for the application-specific system services and for more general business services that are often related to the persistent data maintained. This layering relationship in shown in Figure 1, which is adapted from Cheesman [2]. The general concept is that several different applications (i.e. system services components) can share access to a set of general business services components, that several different user dialogs (e.g. interactive or batch) can use a system services component, and that several different user interfaces (e.g. command line, GUI, Web, etc.) can use a single user dialog component.

The second guiding principle of the course is the *use and adaptation of standard notations and methods*. We adopt well-known object-oriented analysis and design methods and adapt them to the development of component-based system. These include domain analysis, use case analysis, business type modeling, interaction modeling, and design by contract. We also adopt standard notations such as the Unified Modeling Language (UML) [4], Object Constraint Language (OCL) [10], Java, and Enterprise JavaBeans (EJB) [1, 3].

The third guiding principle of the course is a focus on the *development of software families.* Instead of developing single applications, we emphasize the definition of frameworks or software product lines. Such a framework separates the common functionality of the family (sometimes called the *frozen spots*) from the variable aspects of the family (sometimes called the *hot spots).* In component-based

applications, the framework consists of the overall architecture and a set of common components. The hot spots are represented by points at which custom components can be plugged into the system.

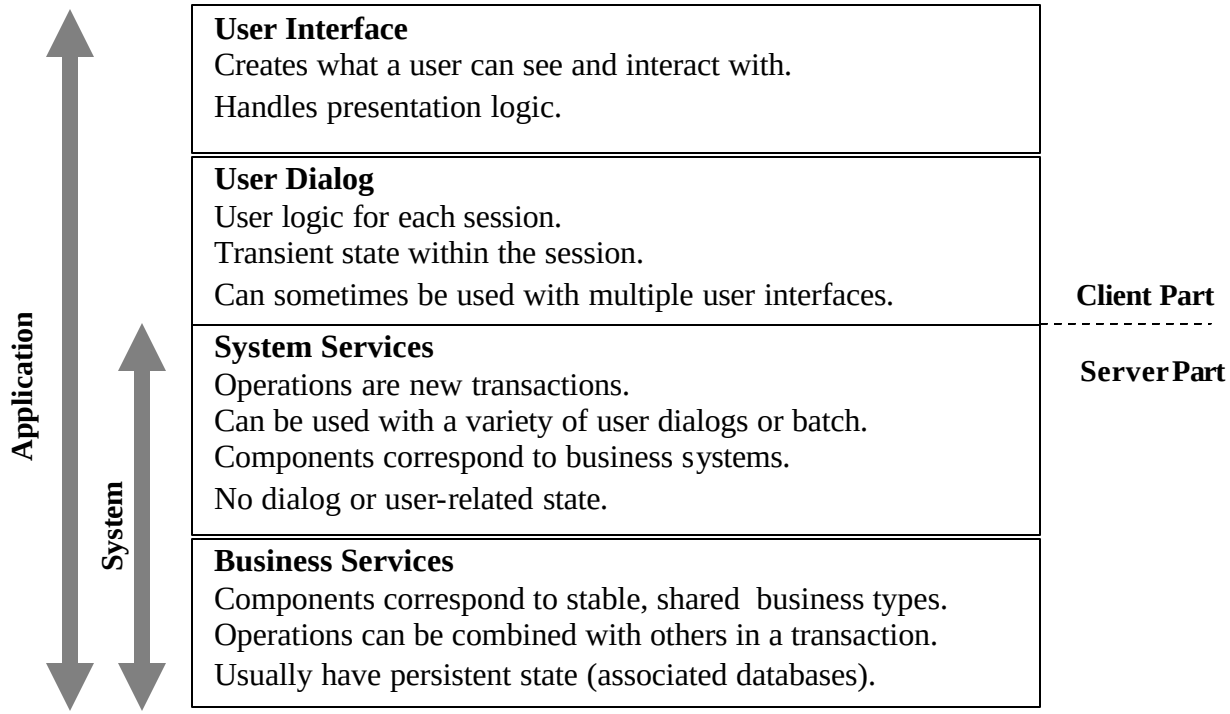

Figure 1. Architectural Layers

3. COURSE STRUCTURE

The course is about component methods and technologies. To be successful, students must learn how to subdivide large systems into reusable and replaceable components. The course is divided into three parts: component concepts and requirements definition, specification, and component implementation.

3.1 Component Concepts and Requirements Definition Phase

The component concepts and requirements definition part of the course takes about four weeks in a typical three-credit semester course. Its objective is to introduce students to the basic component concepts and the methods and artifacts of the requirements definition phase. Requirements definition involves analysis of the problem domain and capture of user requirements for a new system.

First, the course introduces the basic component concepts. Readings for students include selected chapters from Cheesman [2] and Heineman [5]. This study examines a layered architecture that is used throughout the remainder of the course (Figure 1). At the end of this study, the students should be able to answer the following questions: What is component? Why do we need components? What are the basic elements of a component model? What are the differences between a component and an object? What are the characteristics of a component? What are the component design layers?

After introducing these concepts, the course then examines the two steps of the requirements definition phase—*domain modeling* and *use case modeling*. Domain modeling seeks to build a conceptual model of the proposed system's problem area; it

often begins with a grammatical analysis of the system's description in an effort to discover the candidate "classes" (the nouns) and their relationships [8]. Next, the course looks at how to specify the user's view of the system functionality in terms of use cases [2]. Many use cases are identified by analyzing the business processes of the system; others are found by analyzing how the various concepts and relationships in the domain model are created, deleted, or updated. A success scenario captures the primary sequence of interactions of some user role for some major event. For this event, sequences of interactions resulting from errors or user choices are captured in alternative scenarios. Collectively, the scenarios for an event form a use case. The example used in this study is a "course registration system" described in the next section.

Students are given several assignments during the course. The assignments are all related to a case study of a "library circulation management system" which is analyzed, designed, and built by student project groups. The assignments are designed to satisfy the objectives of each part incrementally. Since most students are familiar with the library circulation process, they should be able to carry out the assignments without spending excessive time in learning the problem domain.

The first assignment is to develop a domain model and a use case model for the library system by using the given requirement description. The course briefly examines UML as a notation for expressing the domain and use case models. A UML tool such as Rational Rose is used to complete the assignments.

3.2 Specification Phase

In the second four weeks of the semester, the course examines the specification phase. The objective of this part of the course is for students to learn to divide a system into components and to generate a component architecture and a set of component specifications. The textbook used here is Cheesman [2]. After completion of this study, the students should be able to apply the specification methods to the library system assignments.

The component specifications provide an external view of a component's functionality. There are three stages of the specification phase—component identification, component interaction, and component specification.

The goal of the *component identification* stage is to create an initial set of interfaces and component specifications hooked together into a first-cut component architecture [2]. The component identification stage takes the domain model and the use case model as inputs. For each use case, an initial system interface is defined. The operations on the interface are the major steps in the use case. From the domain model, the course shows how to transform the conceptual domain model into a business type model that gives design information. A key step in this process is the identification of the core business types; these are the concepts that emerge as being independent business-related entities as we analyze the type model. For each core type, the specification method defines a business interface to manage the core type and its related subordinate concepts. The initial component architecture thus consists of the system interfaces and business interfaces and their relationships.

After the study of component identification, the students are given the assignment to build the business type model and the initial component architecture of

the library system based on the domain model and use case model constructed previously.

The next stage is the *component interaction* stage, which is to determine how the components work together to deliver the required functionality [2]. The approach is to decide how to implement the operations on the system interfaces by sequences of interactions with the various business interfaces. This helps the designer discover what operations are needed on the business interfaces defined in the business type model.

The final stage is *component specification*. In this stage, the detailed specification of the operations and constraints takes place [2]. It specifies the interactions between the component object performing the operation and other component objects that are required to complete the operation. It is also necessary to specify the constraints that need to apply to the operations. An interface information model is introduced to enable the definition of these interactions and constraints. The general approach taken is design by contract [2, 7]. In this approach preconditions and postconditions are defined to give the meaning of the various operations on an interface in terms of the information model. Invariants can also be defined to give the constraints on the integrity of the interface information model. The course briefly introduces the OCL as a notation for expressing those constraints [10].

The component specification stage completes the specification of the system. The component architecture consists of a set of system and business interface specifications and how these interfaces interact with each other.

While introducing these three stages, the instructor uses a "course registration system" as a case study to analyze how to build a business concept model, select core types to build business type model, and define the system interfaces and interface information model for the each system interface. Typically all system interfaces will be collected into one system-level component and each business interface will define a separate component; however, business interfaces may need to be combined onto one component or refactored into several components to achieve an appropriately sized component to meet the pragmatic requirements of the system.

At the end of this part of the course, the instructor gives the students the third assignment, which is to complete the operations of the business interfaces and refine the component architecture of the library system.

3.3 Component Implementation

In the previous two parts of the course, students learned how to design a component in theory. In this part, students are required to implement the components they designed by using a particular technology. There are several component technologies, the primary ones being Microsoft's COM+, the CORBA Component Model, and Sun's Enterprise JavaBeans (EJB) [2].

For this course, the instructor uses EJB as the implementation technology. The students are required to master the fundamentals of the EJB technology sufficiently to map the component architecture designed previously to EJB. The textbook used in this part is Deitel [3].

Enterprise JavaBeans is a software component model for developing and deploying enterprise-wide business applications. Enterprise beans are deployed and executed in an EJB container, which exists on an EJB application server. The beans are reusable and shareable components on a server that can be remotely accessed by a client program. EJB is thus suitable for building distributed, reusable systems [1].

The course introduces the three types of enterprise beans: session beans, entity beans, and message-driven beans. Students are required to know the basic structure of each bean and in which circumstances each bean type can be used.

EJBs are a part of Sun's Java 2 Enterprise Edition (J2EE) platform, which is designed to provide a multilayer distributed application model [6]. The architecture of J2EE is shown in Figure 2 (which is adapted from Kassem [6]).

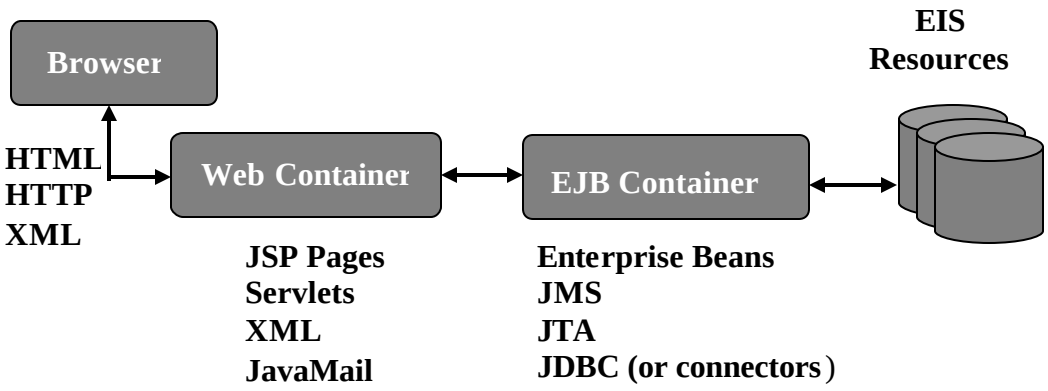

Figure 2. J2EE Architecture

The course uses the course registration system case study. This case study uses HTML in a browser, JavaServer Pages (JSP) and JavaBeans in a Web container, and Enterprise JavaBeans in an EJB container. The case study uses the Java DataBase Connectivity (JDBC) library to access a relational database. Cloudscape is used as the database system.

The important issue in the implementation part is how to map the application architecture layers to EJB. There are several possible mappings. The instructor chose a hierarchical mapping approach as shown in Figure 3. Entity beans operate as business objects and represent data in the database. In the example, entity beans are used to encapsulate database tables. Session bean are used to perform the various processes of the business. Stateless session beans, which provide business methods but do not maintain conversational states, are used to implement the managers for the business components and system components. Stateful session beans, which involve many interactions with clients, can be used to implement the user dialog software level, in which conversational state would be stored. Alternatively, the user dialog could be implemented with Web container code such as JSP scripts or servlets.

Students are not only required to know how to map the architecture to EJB and to code EJB programs but also need to be familiar with the deployment process in the EJB development environment. The course exercises use the J2EE deploytool and the Cloudscape database.

As a fourth assignment, the student project groups are given three weeks to implement the library system by mapping their earlier designs to EJB.

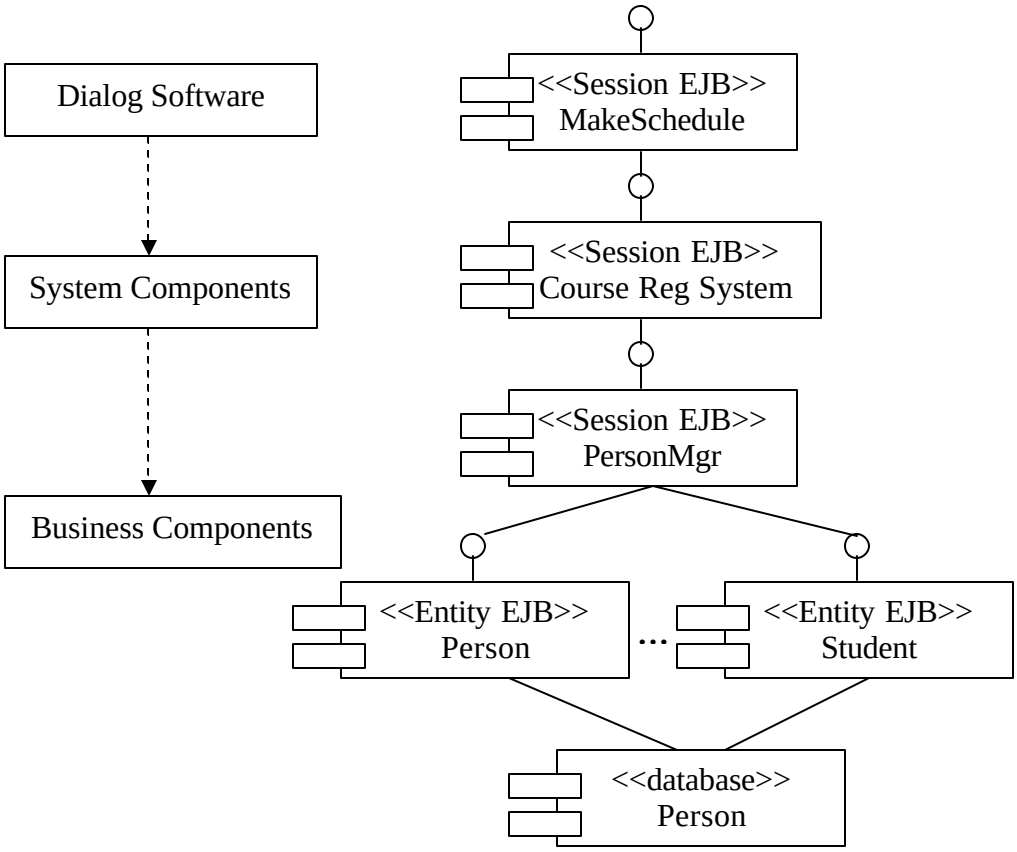

Figure 3. Architecture Mapping to EJB

4. CASE STUDY

A course registration system for a college is used as a case study throughout the course. With this system, a student may register for classes. Once given access, the student may select a term and then build a personal class schedule from among the classes offered that term. A student may add and delete classes from the schedule. The system passes the information about the student's schedule to the tuition billing system. An instructor may use the registration system to print a listing of the students in his or her class. The administrator may maintain student and instructor lists and course information.

After analyzing the requirements, the domain model and the use case model are built. Following the specification workflow, business type model is created during the component identification stage. The business type model for the course registration system is shown in Figure 4. From the business type model, by interaction modeling and component specification, the final component architecture can be specified as shown in Figure 5.

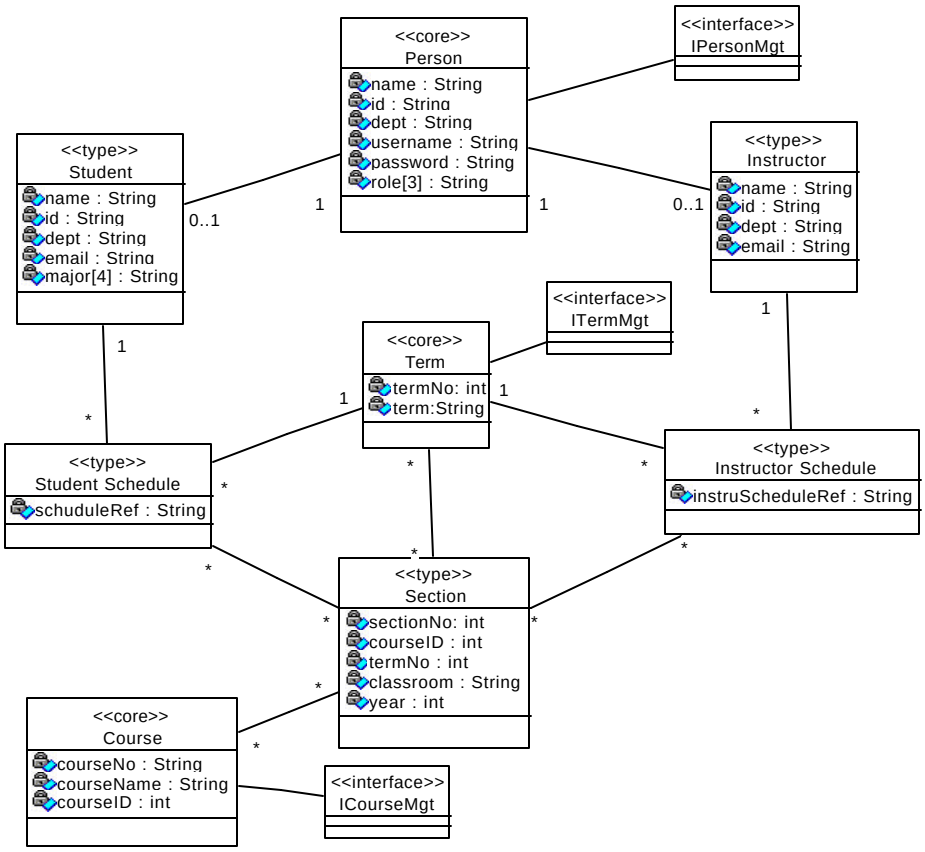

Figure 4. Business Type Model for Course Registration System

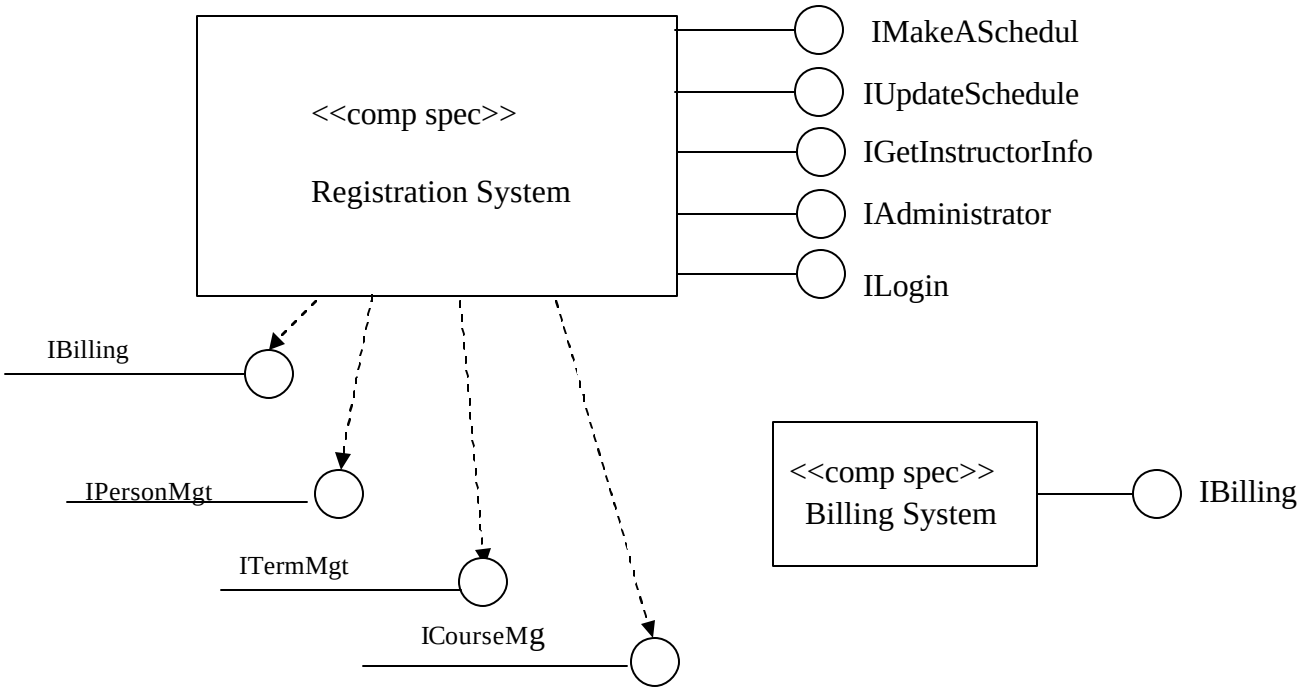

Figure 5. Component Architecture for Course Registration System

5. EXPERIENCES AND CONCLUSION

This course was offered by the first author for the first time in the Fall 2002 semester to a group of 25 graduate students at the University of Mississippi. At the end of the semester, the instructor gave the students an anonymous survey. Of the 24 students responding, 9 reported being "very satisfied" with the overall course, 4 "somewhat satisfied", and 10 "satisfied". In terms of relevance to their future careers, 7 considered the course "very relevant", 8 "somewhat relevant", and 8 "relevant". On their knowledge of component design methods, 12 considered their knowledge "excellent" and 11 "good". On their knowledge of the EJB technologies, 9 considered their knowledge as "excellent", 11 "good", and 3 "marginal". The students also gave suggestions and comments about the course. Most of the student feedback focused on the difficulty of learning the necessary EJB programming and use of the deploytool in the short time at the end of the semester.

The instructor noted that several of the groups abandoned their designs in a crash effort to get something working. Part of this was a result of problems learning the technology while trying to use it to build a small system in the last few stressful weeks of a semester. However, part of the difficulty was a result of designs that were unrealistic with respect to the actual implementation technology.

Based on the student suggestions and the experiences from teaching, the instructor is considering the movement of the introduction of the basic EJB technology to the first part of the course. It is also likely that one to two small EJB assignments will be given to help the students learn to use EJBs. Students will then be familiar with the technology before beginning the final project and their designs will be better informed by a practical understanding of the implementation technology. A possible negative consequence of this is that the students will design to the EJB functionality rather than to the more general concept of component used in most of the course and that the general design methods will be pushed too late in the semester.

Because of lack of classroom time, the instructor gave little attention to several topics originally intended to be included. The design by contract methods and OCL were only discussed superficially. There was no direct discussion of product line (framework) methods. There was also no discussion of the Web container technologies such as JSP. The instructor hopes to reorganize the content to give more attention to framework design and design by contract methods. The plan to include the JSP technologies was unrealistic and will not likely be pursued in a future offering of the course.

In summary, this course was designed to enable advanced undergraduate and graduate students to grasp the concepts of components and the design and implementation of component software. This objective was reasonably well accomplished at the end of the semester. The course brought students a new perspective on software engineering.

ACKNOWLEDGEMENTS

The development of this course is supported, in part, by a grant from Acxiom Corporation titled "The Acxiom Laboratory for Software Architecture and Component Engineering (ALSACE)." It is also supported by the Department of Computer and Information Science and the School of Engineering at the University of Mississippi.

REFERENCES

[1] D. Blevins. "Overview of the Enterprise JavaBeans Component Model," in G. T. Heineman and W. T. Councill (editors), *Component-Based Software Engineering: Putting the Pieces Together*, Addison Wesley, 2001.

[2] J. Cheesman and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*, Addison Wesley, 2001.

[3] H. M. Deitel, P. J. Deitel, and S. E. Santry. *Advanced Java 2 Platform: How to Program*, Prentice-Hall, 2002.

[4] M. Fowler and K. Scott. *UML Distilled*, Second Edition, Addison Wesley, 1999.

[5] G. T. Heineman and W. T. Councill (editors). *Component-Based Software Engineering: Putting the Pieces Together*, Addison Wesley, 2001.

[6] N. Kassem and the Enterprise Team. *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*, Addison Wesley, 2001.

[7] Y. Liu and H. C. Cunningham. "Software Component Specification Using Design by Contract," *Proceeding of the SouthEast Software Engineering Conference*, Tennessee Valley Chapter, National Defense Industry Association, April 2002.

[8] D. Rosenberg and K. Scott. *Use Case Driven Object Modeling with UML: A Practical Approach*, Addison Wesley, 1999.

[9] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, 1998.

[10] J.Warmer and A. Kleppe. The *Object Constraint Language: Precise Modeling with UML*, Addison Wesley, 1999.

BIOGRAPHIES

H. Conrad Cunningham is Chair and Associate Professor of Computer and Information Science at the University of Mississippi. His professional interests include concurrent and distributed computing, programming methodology, and software architecture. He has a BS degree in mathematics from Arkansas State University and MS and DSc degrees in computer science from Washington University in St. Louis.

Yi Liu is a PhD student at the University of Mississippi with an interest in software engineering and artificial intelligence. She has a Master's degree in computer science from Nanjing University in China.

Pallavi Tadepalli is a PhD student at the University of Mississippi with an interest in software engineering. She has a BE degree from the Government College of Engineering in India and is a candidate for the MS degree in computer science from the University of Mississippi.

Mingxian Fu is a part-time Research Associate at the University of Mississippi with an interest in software and Web-based development. She has a BS degree in mathematics from Yunnan University in China and a recent MS degree in computer science from the University of Mississippi.