Teaching Enterprise Application Development: Strategies and Challenges

David Whittinghill
Purdue University
401 N. Grant Street
West Lafayette, IN 47906
davewhit@purdue.edu

Kyle Lutes
Purdue University
401 N. Grant Street
West Lafayette, IN 47906
kdl@purdue.edu

ABSTRACT
Enterprise application development requires a skill set that is broader than that provided by traditional programming courses. Enterprise applications are distributed, networked, multi-user, and, in most instances, fairly complex. As such, competent software developers must possess more than the basic understanding of a programming language. Developers of enterprise applications must also possess an understanding of networking fundamentals, network and application security, relational database management systems, concurrency, and application deployment and scaling. They should additionally be knowledgeable with the mechanics of at least one specific application platform, for example J2EE or Microsoft’s .NET. This collection of competencies tend not to be taught in the software development curriculum as a collective whole, rather they are touched upon in many different courses in the curriculum. Though this traditional approach is more or less successful in creating adaptable skills for students who are then positioned to learn application domain-specific technologies and concepts, this approach is likely to fail in providing in-depth knowledge of the principles, patterns, and techniques used for developing these types of large enterprise applications. The acquisition of this knowledge must therefore be obtained in either an on-the-job or self-taught fashion.

Our department has been addressing these issues for over ten years via a software development course that uses a variety of technologies including COM/DCOM, .NET, and J2EE. Our expressed goal is to educate our students with the knowledge required to design and implement enterprise applications.

In this paper, we present the goals and history of our course, the framework we have used for teaching the subject, as well as the challenges that can arise in administering the associated coursework. The observations are based on the two authors’ collective experience teaching enterprise application development in a university setting.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: General; D.2.8 [Software Engineering]: General

General Terms
Design

Keywords
Enterprise application development, Education, Teaching

1. INTRODUCTION
Modern information technology curricula are typically oriented toward principles and technology fundamentals[18]. One result of maintaining this orientation is that, by and large, instructional topics in the university setting tend toward a division-of-labor perspective in which, for instance, general programming, algorithms, databases, networking, etc. are taught as discrete courses. For decades this has been the dominant model in computer science and information technology education[6][17] and, as evidenced by the existence of our contemporary technology-saturated, digital, connected lifestyle, it can safely be regarded as successful.

This success has spawned new challenges. The proliferation of excellent technology and technologists has enabled the creation of new classes of software that exist as aggregates of other technologies[19]. Though distributed architectures are not a new idea[1], conceptualizing software as an aggregation of components and services is an idea that has gained greater traction as networked computing resources have become more ubiquitous[4][20]. In the past, the dominant assumption was that software should be designed to operate as a stand-alone, start-to-finish resource that existed on a user’s desktop or within an organization’s network environment[1]. Component-based software development (CBSD) methodology was an attempt to increase modularity and reusability of software by characterizing software as “an independently deliverable set of reusable services”[21]. Today this concept is further advanced as Software as a Service (SaaS) in which software is conceptualized as a collection of always-on networked services that each perform only one particular function or set of functions[3].

The shift from stand-alone installed client software, to service-oriented, component-based software is not trivial. This paradigm imposes more demanding operational expectations and makes different assumptions about the deployment environment[8]. Applications must be equally avail-
able to an unknown number of users, simultaneously, continuously, and must do so without access to a shared memory space (and therefore without access to the application's state). Further, they must do so in an application-agnostic fashion and with minimal overhead. Addressing these sorts of concerns are beyond the scope of the traditional information technology core curricula but are foundational for successful enterprise application development (EAD) and SaaS.[8]

Despite the growing presence of enterprise applications in the computing field it is unclear, given current priorities in computing education, how EAD-centric skills will be taught to new generations of developers. As the field of computing grows in depth and breadth a challenge arises in determining how much of that breadth and depth can be fit into the finite space of the four-year university degree[22]. EAD, when it is taught, is typically taught as an elective; even then, due to competition for students' time, it can be difficult getting new EAD courses established[7]. Conversely, in some academic programs EAD is seen as a unifying capstone course that challenges students at the end of their undergraduate careers to apply their accumulated knowledge to difficult enterprise implementations[7][9][11]. Fortunately, an IEEE Services Computing Curriculum Initiative (SCCI) Congress has been formed describing the underlying objectives, concepts, and future directions of service-based computing[23] which will benefit not only service-based development but enterprise application development as a whole; that this Congress was convened is indicative of the broader acceptance and intellectual momentum of the subject domain and helps justify the creation of EAD coursework and the elevation of its prominence within the undergraduate computing curriculum.

2. OVERVIEW

Enterprise application development (EAD) can be a challenging topic for students to learn and for instructors to teach as it requires, at the outset, a working knowledge of programming languages, application development, database administration, and networking. We describe our format and experiences teaching EAD, the challenges we have encountered, and the strategies we have employed to address them.

3. HISTORY

3.1 Department History and Goals

The Department of CIT at Purdue, unlike many traditional information technology curricula, has focused upon the application level of computing with its research efforts centered around the successful application and deployment of current technologies. Students in CIT choose from one of four specializations: Software Development, Systems Integration, Data Management, and Computer Networks. Though theory is taught in the CIT department, the hands-on application of technology is strongly emphasized throughout our students’ undergraduate careers. The department of CIT reaps the benefit of close contact with industry partners who keep faculty abreast of current trends and issues, thus helping to maintain informed course materials that are relevant and realistic. Graduates from the Purdue CIT department typically go on to work in IT-related industries (as software engineers, IS managers, database administrators, network administrators, etc.) or seek graduate degrees upon graduation.

3.2 Reasoning Behind EAD Course

Prior to fall semester 1999, the Software Development area of specialization did not have an advanced programming course as part of its curriculum. Though basic programming concepts such as programming languages, algorithms, and data structures were being taught, the faculty determined that a senior-level advanced course was needed as well. The new course should provide not only challenging programming problems, but an opportunity for students to integrate the entirety of their technical studies and to do so in an area that had high real-world strategic potential. Development in large-scale, multi-tier, multi-user, client/server architectures forces developers to consider issues not seen in single-system architectures such as concurrency control, class library development, tiers, abstraction layers, and session management, among others. A new course, CIT 450 Enterprise Application Development, was created to address these needs.

3.3 History of CIT 450 Enterprise Application Development

CIT 450 was first offered in fall of 1999. From then until fall of 2002, it had been offered each semester after which it is now offered once per year. The course typically has enrollment of between ten and twenty students. Though EAD touches upon all four of CIT’s areas of specialization, the course is administered by the Software Development faculty and thus is oriented more toward students with strong programming skills.

4. CIT 450 ENTERPRISE APPLICATION DEVELOPMENT

4.1 Selected Technologies

Initially, CIT 450 was taught using VB6 with COM/DCOM. The course eventually migrated to C#, .NET and XML Web Services and, more recently, Java. Development platforms such as Sun’s J2EE (Java 2 Enterprise Edition) and Microsoft’s .NET are currently the dominant platforms in which enterprise applications are implemented. Though the two platforms compete for developer mind and market share, currently neither has a clear-cut technical advantage over the other, particularly in the area of service-oriented architectures[10][16]. As such, a challenge in CIT 450 has been to strike a balance between teaching concepts that generalize well while also creating sound implementations that leverage a platform’s particular strengths.

4.2 Pedagogy

CIT 450 meets three times per week, two lecture periods and one lab session. Homework is always writing code (no papers, speeches, presentations, design docs, etc.). Assessments consist of ten homework/lab assignments, ten quizzes, a midterm exam, and a final exam. For lab assignments, a sample domain data set is provided (e.g. real estate properties for sale, students and the classes they are enrolled in, the University course catalog, a song library from an iPod, etc.). Lab assignments build upon one another with each new assignment being an iteration upon
the previous. The lab assignments coincide with each new lecture topic.

For each assignment, students begin with source code applicable to one architecture and refactor to work in a more complex architecture. For example, migrating a single-user application to a multi-user application. This cycle of implementation-refactor demonstrates by example not only how to implement enterprise application but also how assumptions that work well in a single-system programming environment can fail in the enterprise environment.

### 4.2.1 Lab Assignments

Though there are semester-to-semester variations, the sequence below gives an example sequence of the evolution of programming assignments as they progress through the semester.

- **Lab 1.** Single-user desktop application with sequential file I/O is rewritten in OOP-style to separate UI logic from app and data logic [14].
- **Lab 2.** Solution to Lab 1 is refactored to include proper exception handling.
- **Lab 3.** Solution to Lab 2 is refactored to change data storage to use a desktop RDBMS for data storage. DBMS abstraction layers are implemented. Changes to the UI class are not allowed.
- **Lab 4.** Solution to Lab 3 is refactored to use a 2-tier client/server grade RDBMS. Changes to the non-DBMS abstraction layer are not allowed. The dataset is changed to include tens of thousands of records.
- **Lab 5.** Solution to Lab 4 is refactored to add maintenance activity (adds, updates, and deletes, or “who did what and when did they do it”) to the database. Transaction processing is also added to the business layer classes.
- **Lab 6.** Solution to Lab 5 is refactored to handle concurrency issues (e.g. what do you do when two people attempt to modify the data in a single record at the same time?).
- **Lab 7.** Solution to Lab 6 is refactored to remove the business logic from the app and use it to create a binary reusable class library with full IDE documentation.
- **Lab 8.** A new Web UI is developed for the class library created in Lab 7.
- **Lab 9.** An XML web service (WS) is created to expose the methods of the class library from Lab 7 so that it can be called as RPCs. The UI code from Lab 7 is refactored to use the XML WS.
- **Lab 10.** The dataset used in all previous labs is enhanced to include several more tables including a relational entity table. Students enhance the solution to Lab 7 to allow the user to use a single UI form to modify the data that has been stored in several tables.

### 4.2.2 Lecture Topics

Lecture materials consist of a combination of principle-based and implementation-based lessons. Though lectures are biased toward teaching principles, our core philosophy in CIT is one of understanding-by-doing. As such, every attempt is made to teach implementation-specific information about the latest technologies in conjunction with enduring concepts.

**Object Oriented Programming Concepts.** Sound object models are critically important when designing large, complex software systems. We assume students understand the mechanics of OOP, but also examine OOP orthodoxy and the pros and cons of pure OOP implementations. Material on coding to interfaces is also included.

**Proper Exception Handling.** The role of exception handling within distributed, multi-tier systems is discussed. How using exception handling to report system errors without contaminating the data model is also covered.

**DBMS Connectivity Options.** We discuss which database management systems (DBMS) could be used and what are the pros and cons of each. We also examine stored procedures and the advantages and drawbacks they introduce.

**Comparison of Application Architectures.** Studying the evolution of application architectures tells us not only history, but illustrates the cycle of problem-solution-unintended consequence that is the ongoing challenge of system architects and technologists in general. We study the following architectures, the conditions each was meant to address, the problems each solves, and the problems each introduces:

- **Mainframe**
- **Desktop**
- **2-tier client/server**
- **Multi-tier client/server**
- **Web**
- **Service Oriented Architecture**

**Enterprise Design Patterns.** The notion of design patterns as well as several of the more prominent published patterns are discussed [19] [20]. Of particular interest are the Model-View-Controller, Factory, Data Access Object, DAOFactory, and Transfer Object patterns.

**Multi-user Concurrency Issues.** Like most applications, a mechanism exists for preserving application state. In multi-user enterprise environments, guaranteeing that record updates are not lost requires defensive coding strategies that are unnecessary in a single-user environment[13]. A number of strategies are presented and analyzed.

**Binary Reusability via Class Libraries and Third Party Components.** Object-oriented programming languages promised to improve software development by allowing for easier code reuse. However, binary reusable components and class libraries have proven to be more successful. In this unit, we discuss the pros and cons of using components from a third-party vendor. For example, licensing fees, ability to find and correct errors, and vendor longevity are all discussed. Additionally, students develop their own class library component, including self-documenting features that allows the component to be integrated with the IDE.

**Remote Procedure Calls and Distributed Technologies.** As application architectures evolved to become more distributed,
various techniques for marshaling data between application component processes were developed. In this series of topics, CORBA, COM/DCOM, Java RPC, .NET Remoting, and XML Web Services are discussed. The Impedance Mismatch Problem. CIT 450 students have already had several courses in Relational Database Management Systems (RDBMS). CIT 450 students have also had several courses using object-oriented programming languages. In this unit students are taught how the design goals of RDBMs and OOP differ, the problems created by these design goals, and solutions commonly used to overcome these problems. These problems include the differing data types, data redundancy, object state persistence, and performance tradeoffs.

5. PROBLEMS AND ATTEMPTED SOLUTIONS

Teaching enterprise application development is hard [12]. A number of problems, both logistical and intellectual, are of particular concern when teaching enterprise application development. Currency. As is the case in most technical topics, the rate of change in the area of enterprise computing is quite high. What may be perfectly applicable at the time of this writing might be considered quaint and obsolete within only a few short years (or sooner!). This impermanence of the state of technology argues for the teaching of only core, enduring principles at the behest of platform-specific instruction. We concur with the assessments by Cabot et al. [5] and Arora et al. [2] that a mixture of principles and specifics is a successful overall approach as the hands-on nature of the assignments unify the high-level concepts and low-level examples into a more durable understanding of the material. Variation of Student Experience. A problem we have faced when teaching CIT 450 has been that incoming students arrive with different levels of ability. Unfortunately, this is a problem for which we have not found an easy solution. As EAD requires students to have proficiency in programming, database administration and, to a lesser degree, networking, it is uncommon for incoming students to be fully prepared to excel. Our students on average tend to be strong in one area but not all of these areas. For instance, some students have have only a bare minimum of programming while others have been paid programmer interns for a couple of years. We have attempted to address this by 1) reviewing OOP and database concepts early in the semester in an effort to set a base line for all students, and 2) warning students that CIT 450 is a capstone course and that it will be challenging. We have found that while this approach can suppress enrollment it does tend to have a rallying effect on the students who choose to stay. Variation in Student Motivation. A problem that is likely endemic to any senior-level course, our own included, is one of student motivation. While most students take the course because they are genuinely interested in the material, we have come to expect there will be a significant minority of students who are only using the course to plug a hole in their schedule, or perhaps have a post-graduation job waiting for them and are less engaged. Exploring the broader problem of senior-level student motivation is beyond the scope of this article, however we are inclined to agree with others [7] [11] that the EAD makes an excellent senior capstone course that, if it were required, would be an excellent opportunity for students to demonstrate their readiness to graduate. An additional approach that seemed to have increased student engagement was described by Iyengar [9] in which the class created their enterprise system in cooperation with an industry partner for whom the system would be created. Grading. A web enterprise application, if constructed according to a pure MVC design needs at least three servers: database, web, and application. As such, running and evaluating an enterprise application requires that three separate deployments occur on three separate systems. Grading the quality of the class’s completed enterprise applications therefore requires the grader to perform three deployments for each student in the class. Though using virtual machines [15] removes the need for additional separate machines, the time required to do grade an entire class assignment is still considerable. Further, as others have noted [5] it is difficult to find qualified student help; since the course is senior level all of the students who have taken it have graduated, and the remaining students are not yet qualified to grade the material. One solution that has worked reasonably well for small classes was to grade homework during lab sessions however this does have the result that there is then no time left for additional lectures.

6. CONCLUSIONS

Enterprise applications provide an excellent opportunity for undergraduate students to challenge themselves by bringing their accumulated knowledge to bear on an interesting and challenging set of problems. As computing in general becomes more distributed and mobile the lessons of enterprise application development become more and more broadly useful. Teaching enterprise application development imposes logistical burdens upon the instructor such as difficulty in grading and a fast-changing technological environment. A mixture of high-concept and hands-on examples should provide the student with the knowledge to be effective in the job market immediately while being able to adapt to unforeseen changes in the technology landscape. Teaching enterprise application development is inherently difficult and as such there are many unresolved issues surrounding its instruction, but the educational and employability rewards to the student make the required extra effort worth the expense.

7. REFERENCES


