A Hands-On Overview Course for Computer Science and Modern Information Technologies

Lixin Tao and Li-Chiou Chen

Seidenberg School of CSIS, Pace University, White Plains, NY 10606, USA
ltao@pace.edu, lchen@pace.edu

Abstract - This paper describes the logics and experience in piloting a new project-based program overview course to introduce the fundamental computer science concepts, methodologies and technologies underpinning the latest information technologies for students who have just completed CS2 or the equivalent. The course focuses on recurring great computing ideas, motivates computing students with the modern real-world challenges and their innovative solution technologies based on creative application of the great ideas, and shortens the prerequisite chains for subsequent computing courses to support more current and flexible computing curricula. The assessment data indicate that students who have taken this program overview course are better motivated and prepared for follow-up courses and real-world IT challenges.

Index Terms - Computing program overview course, Project-based education, Server-based computing, Shortening course prerequisite chains.

INTRODUCTION

The enrollment of the USA undergraduate computer science programs has dropped significantly in the recent years. As reported by CRA (Computer Research Association) [6][2], the number of newly declared computer science (CS) majors declined 33% from fall 2002 to fall 2004. Minority students continue to be underrepresented, e.g., only 3.1% of CS Bachelor degree recipients are African-American students [6][2]. Apart from the burst of the dot-com-bubble and the IT out-sourcing, there are two major reasons that have attributed to the decline of computer science enrollment: (1) the lag between the knowledge scope of our current computer science curricula and the expectations of the IT industry; (2) the current waterfall teaching model.

I. Knowledge Lag Problem

Since early 2000s the IT industry has adopted the service-oriented computing model. As a generalization of the Web and distributed computing technologies, Internet business services [23] (for which Web service is one of the particular implementation techniques) are provided on servers through the Internet for heterogeneous client systems to consume. An Internet business service abstracts specific business logics and their implementations, the server IT infrastructure, and the expertise for maintaining the server resources. The clients of such services are typically software systems that can consume the services with remote system integration. Credit card processing is a typical Internet business service provided by major financial institutions. New Internet business services are typically implemented by integrating existing services, and the XML technologies are the foundation of data integration across heterogeneous IT systems. Internet business services promote specialized and collaborated computing as well as support competitive global economy. Web service is a particular implementation technology of Internet business services, and service-oriented architecture (SOA) specifies the software architecture based on service integration. The service-oriented computing is based on networking, the client-server and thin-client architectures, and the Web architecture which is still the foundation of the fast-growing e-commerce and e-society.

As the top-level abstraction, each Internet business service is implemented with server-side software component technologies like EJB [22] and .NET [24]. A software component is a software module that has well-defined interfaces and can be individually deployed. A software component typically implements specific business logics with multiple objects, and the common server infrastructure functions, like component life cycle management, thread pooling and synchronization, data caching, load balancing and distributed transactions, are factored out into a component container which is basically a software framework interacting with the components through pre-declared hook and slot methods. Since early 1990s the software component based software engineering has become the mainstream IT industry practice. In 1995 The Department of Defense mandated that all its projects must be based on software components.

Based on the above discussion we can see that over the last two decades the concepts of abstraction and divide-and-
conquer have been recursively applied to higher-level software modules/systems, from objects to software components and to Internet business services; the knowledge base for server-based computing is a superset of that for client-side computing and introduces many new challenges not properly covered by the current curricula; and the dominant server-based computing IT technologies are based on sound and recurring concepts and methodologies that must be integrated in computer science curricula to prepare students for the current and future IT challenges.

But many of our computer science programs are still struggling with effective teaching of objects and have weak coverage of server-side computing. Most of the concepts and methodologies mentioned above are either only covered in elective senior courses, or weakly covered, or totally missing in the current curricula. Our students need early introduction of the fundamental modern computing concepts so they can have a clear roadmap and motivation for their programs and be well-prepared for the competitive global job market. ACM Computing Curricula 2001 correctly introduced the net-centric knowledge area to address the above knowledge gap, but most computer science programs have not properly integrated it into the curricula due to limitations of faculty expertise and resources.

II. Waterfall Teaching Problem

Most of the computer science curricula today are still based on ACM Computing Curricula 1991 that reflected the IT technologies at that age, with limited coverage on server-based computing. The topics are covered in their waterfall order specified by the existing prerequisite chains. Even though the fundamental concepts in these curricula are still the foundation of today’s technologies, many important concepts and skills are scattered in many senior courses which cannot be taken earlier due to the strict course prerequisite requirements. For example, a typical engaging programming project today involves graphic user interfaces, database and networking. To make user interface responsive, multi-threads are needed. But most of the current curricula introduce networking programming and database programming in senior courses, and introduce multithreading in an operating system course in the 6th semester. As a result the instructors are limited in what kind of projects they can use to engage the students, and the students have limited opportunities in practicing the important skills. To resolve this problem we need to switch away from the current waterfall teaching model and greatly shorten the current deep course prerequisite chains.

We advocate an in-depth lab-based computer science program overview in the junior year with emphasis on the fundamental and recurring concepts and skills underpinning the modern computing technologies. Such hands-on program review can be in the form of a new “Introduction to Server-Based Computing” course for students who have completed CS1 and CS2, or be integrated into the relevant existing courses as complement course components reflecting the latest technology trends. Such a computing overview course can also be adopted by Information Systems programs and Information Technology programs since students in these programs also need to understand the latest computing concepts and technologies to address the business problems, or deploy and maintain business solutions on servers.

RELATED WORK

Most computer science programs introduce multi-threading in a junior operating system course, database programming in an elective senior database course, event-driven programming in an elective course on graphic user interfaces, and networking programming in a senior networking course. In response to ACM Computing Curricula 2001, many computer science programs have introduced Web development courses. But these courses focus on developing Web sites instead of the fundamental concepts and methodologies behind the tiered Web software architecture. For example few of these courses address the server scalability problem or cover the application server tier, which factors out the common server infrastructure from business logic implementations. As an example from the top universities, over the last five years MIT introduced a junior-level course “6.171 Software Engineering for Internet Applications” [19][20] which covers a subset of the topics that we cover in this project, but this course focuses on motivations and software design, and it depends on student self-study to fill in a lot of gaps. For example XML technologies are used in exercises for textbook’s Chapter 2 but are not introduced in the textbook. This could lead to difficulty in porting the course to the computer science programs with limited domain experts and resources. As another example from the average universities, during 2006-2007, the Computer Science Program of the Loyola University in Chicago offered a very popular new course “COMP388/488: Server-Side Software Development” [21] for undergraduate students who have just completed courses Data Structures and Algorithms and Introduction to Computer Systems (in C). This course basically covers the Web technology knowledge topic in our new course with focus on JavaServer Pages [22] and Jakarta Struts [22]. This course is also a proof-of-concept that undergraduate students can grasp the fundamental concepts and skills behind server-based computing.

METHODOLOGY

I. Iterative Teaching Model

Education research [25][2] indicates that iterative teaching is more effective than sequential/waterfall teaching. Problem solving depends on many knowledge topics, and these topics often have tight inter-dependency among them. Waterfall
teaching will limit students’ view of the complete education program and the type of engaging projects that the students can work on to enhance the learning process. Therefore we suggest that a new “Introduction to Server-Based Computing” course be introduced immediately after CS1 and CS2 as the end of the first iteration of the computer science education so the students can have a global perspective of their education programs and master the basic hands-on problem-solving skills during their junior year. This course provides practical overview for advanced data structures (vector, hash table, stack and queue), operating systems (Linux and Window shells, processes vs. threads, thread synchronization, critical region, and referential locality underpinning memory hierarchies), databases (database management basics, SQL language basics, database programming), networking (TCP/IP basics, socket-based networking and their software framework abstractions), component-based software engineering (component reuse and integration, component life-cycle management, factoring common infrastructure into component containers), XML (syntax specification of XML dialects, XML instance document parsing and validation, and language transformation), Web technologies (Web architecture, XHTML basics and HTTP, dynamic Web page creation and session management), Web services, and grid computing (parallel and distributed computing, grid services). This course will be followed by or in parallel with the second iteration consisting of courses including data structures and algorithms, operating systems and architectures, networking, as well as many elective senior courses. Due to this new course in the first iteration, the courses in iteration 2 can be taught in more flexible orders and with more depth and hands-on projects. For example students can work on a software engineering project without having taken a database course. The capstone course normally makes the 3rd iteration allowing students to integrate the learned knowledge topics in problem solving and learn new knowledge/technologies with limited instructor assistance, which is important in developing life-long learning abilities.

Our approach is similar to the breadth-first approach described in ACM Computing Curricula 2001 [5]. But the breadth-first approach is usually implemented as an extra first course before CS1 to provide a program overview. Due to its timing it cannot cover advanced concepts or hands-on projects.

II. Hands-On Problem-Solving

Education research [2] indicates that motivated students are more effective ones. Today the traditional programming applications like calculators cannot motivate students much. Early on the students expect to see the connection between what they learn and what is happening in the real world, and how to apply their learned knowledge in problem solving. Therefore we integrate weekly hands-on projects as well as some design projects, like the design of an RSS framework for news feeds, into the teaching of each topic in the proposed “Introduction to Server-Based Computing” course.

III. Scaled-Down Sample Projects for Real Technologies

One major reason that why we can cover so many important modern computing concepts and skills in a single course or a small set of course modules is because we use a set of well-designed sample projects to lead the learning process. For each major topic we have carefully designed a set of sample projects for students to study, modify, and enhance. For example it is not practical for students to learn the complete servlet or EJB container design and implementation within a week, but students can study a simplified software framework project to see how common functions can be factored out to a container and how the container can control the life cycle (instantiation, initialization, serialization, deserialization) of objects running in the container. This course is for providing program and technology overview, therefore its focus is for students to master the fundamental concepts and problem-solving skills. We purposely avoid industry-quality technology implementations because more than 80% of their code is for perfection or extra features not essential to understanding the fundamental concepts.

PACE EXPERIENCE

In 2006 Pace introduced a new lab-based course “CS612 Concepts and Structures in Internet Computing” for introducing the fundamental concepts and skills for server-based computing to new computer science graduate students as well as selected undergraduate computer science students who have just finished CS2. The introduction of this course was one of the strategic steps for Pace University to attract more quality undergraduate students to its computer science program. The course has been offered for four years at Pace University with great success.

Pace’s course “CS612 Concepts and Structures in Internet Computing” closely followed the author’s textbook draft “Introduction to Server-Based Computing” [4] and has the following organization in its spring 2010 offering. A VMware [7] Ubuntu virtual machine and a Windows XP virtual machine with pre-installed and pre-configured course software were used to provide the students with the same lab environment on any Windows or Linux PC at university labs or at student homes.

Introduction to Server-Based Computing

1) Introduction (1 week)

Topics: Challenges of enterprise computing: concurrency, scalability, atomic transaction, system integration, data integration, data persistency, complexity encapsulation. Fundamental computing paradigms: sequential, object-oriented, event-driven,
and multithreading (as abstraction of real-world parallelism and a methodology for separation of concerns). Fundamental methodologies: abstraction and divide-and-conquer.

**Labs:** Installation and configuration of *Java JDK, Apache Tomcat, MySQL, Eclipse*; setting environment variables *PATH* and *CLASSPATH*; learning organization, compilation and execution of classes in packages; deploying two sample Web applications Bookstore1 (*Servlet*) and Bookstore2 (*JSP*) in *Tomcat*; creating database and tables for the bookstore sample projects with prepared *SQL* files.

2) **Object-Oriented Computing Paradigm** (2 weeks)

**Topics:** Concise review of object-oriented computing, including packages, interfaces, class inheritance, polymorphism, exception handling, I/O, multithreading and synchronization, vector and hash table. (Textbook Appendix A provides reviews for students who need more warm-ups.)

**Labs:** One or more two-page source code examples for each topic, starting from *Hello World* and growing incrementally in complexity. Compile and run each demo project on command-line and in *Eclipse*.

3) **Client-Server Computing and the Internet** (1 week)

**Topics:** IP addresses, domain name, DNS, ports, TCP/IP fundamental concepts, sockets as communication abstraction, client-server architecture.

**Labs:** A four-page client-server application for calculating disc areas by the server. Modifying the application to implement a server-based accumulator. Enabling concurrent clients. Using *ThreadGroup* to limit the number of concurrent clients. Running the applications among PCs in a lab and over the Internet.

4) **Event-Driven Computing Paradigm and Human-Computer Interaction** (1 week)

**Topics:** Events, event sources, event consumers/listeners, interaction among event sources and their listeners; event-based system integration; database trigger concept; Java user interface components: text field, button and label; Java layout managers; Java listener interfaces for keyboard and mouse events.

**Labs:** Java GUI programming with text fields, buttons, labels and panels. Implementation of a simple generic event-driven programs based on Java interface *Observer* and class *Observable*; design and implementation of interface *MyObservable* and class *MyObservable* with a *Vector* to duplicate the corresponding Java library implementation.

5) **Component-Based Software Engineering** (2 weeks)

**Topics:** Software framework for factoring out the implementation of common functions of a class of applications; component containers vs. components; hook and slot methods; component life cycle management; thread pooling and synchronization; component-based networking; components vs. objects.

**Labs:** Study a simple *Object-Client-Server-Framework (OCSF)* with four source code files for implementing client-server applications. Implementation of a simple command-line instant messenger based on *OCSF* to see how socket-level details are abstracted into the software framework and developers only need to focus on business logics.

6) **Data Persistence** (1 week)

**Topics:** Database fundamentals. Database server machines, database management systems, databases, and relational database tables. SQL language basics. Primary key vs. foreign keys. JDBC and ODBC. Database atomic transactions.

**Labs:** Use a one-page Java program to create a table, insert data, query data, and drop the table on *MySQL*. Modify the code to repeat the same experiment on *Microsoft Access* by creating a *Windows* system data source. Learn *MySQL* administration basics.

7) **Data Integration** (1 week)

**Topics:** Separation of data structure from data presentation; XML dialect specification with DTD, *XML Schema* and namespaces; data validation and parsing with SAX and DOM; data transformation with XSL.

**Labs:** Exercise on all aspects of XML technologies with author’s XML demo applications; specification of news data structure with *XML Schema*; XML news data syntax validation with SAX; XML news data transformation into HTML files for desktop PC and cellular phone presentations.

8) **Web Computing** (1 week)

**Topics:** HTML form and its various supported fields; URL structure; *HTTP GET* and *POST* requests; the four-tiered Web architecture; the Web server architecture; role of application servers; session management with cookies, hidden fields, URL rewrite and server session objects; CGI vs. Java servlet/JSP Web applications; Web server configuration and security.

**Labs:** Observe HTTP request/response messages; study simple Web applications illustrating session...
management; study the design and implementation of a Java Web Server with 20 pages of source code; add Perl CGI support to the Java Web Server; develop simple servlet and JSP applications for a guest-book application with guest information maintained in a MySQL database.

9) **Web Services and System Integration** (1 week)

**Topics**: Thin-client computing; service-oriented computing and architecture; Web services for integrating heterogeneous information systems across the networks; describing service interface methods and method invocation data with XML dialects WSDL and SOAP; the proxy design pattern; fast transformation of legacy applications for implementing Internet business services; tradeoff of client-side and server-side computing with Ajax; security issues with Web services and client-side computing.

**Labs**: The author’s two-hour hands-on introduction to Web services on both Java platform’s Tomcat with the Axis toolkit and Microsoft Visual Studio .NET in C#, which also shows the cross-platform Web service invocation from Java to C# and vice versa. The Google Web Toolkit is used for introduction to the Ajax framework.

10) **Distributed and Parallel Computing** (1 week)

**Topics**: Objectives and methodologies of parallel and distributed computing; grid computing; concurrency vs. parallelism; data mapping for distributed memory architectures; resource management and load balancing; introduction to industry standard Message-Passing-Interface (MPI) [17] for parallel computing on PC clusters.

**Labs**: Introduce to a pure Java grid toolkit that has small footprint and can be used to transport both data and binary Java code to PCs in a network to run Java applications in parallel.

11) **Conclusion** (1 week)

**Topics**: Mapping of follow-up courses to the course concepts; mapping of the current IT technologies to the course concepts; identification and reiteration of fundamental computing paradigms and problem-solving methodologies that underpin the past 50 years’ computing innovations.

The dependency among the first ten course modules is described in the following Figure 1.

This course has been scheduled for both fall and spring offerings in computing labs. So far over 80 undergraduate and graduate students, who have varied computing background, have completed this course. In addition to the guided lab sessions integrated in the lectures, students need to complete one mini-project assignment each week for the week’s topics and complete one course project “Logic Design of an RSS News Feeds Framework” at the end of the course. The course project enables the students to apply their learned knowledge in XML data modeling, Web service system integration, and multithreading and event-driven programming into the design of the news feed server and reader application. All the students completed the course successfully.

**Figure 1 Course component dependency**

**CONCLUSION**

A new hands-on overview course for computer science and modern information technologies have been developed at Pace University, and its offerings in the last four years proved the effectiveness of the course in engaging students in computing programs, shortening computing program prerequisite chains, and introducing the latest IT technologies and challenges into computing education and research. The Pace University is ready to share the complete courseware with universities that are interested in adopting a similar course.

**REFERENCES**


Qian, K; Tao, L and Fu, X. “An Active/Cooperative Role-based Open Source Digital Labware for Information Forensics,” Education, IEEE ICALT 2007, 07/07


VMware, http://www.vmware.com


