A Web-Based Application Development Course for the Computing Curricula 2001/NC3 Track

T. Andrew Yang

Computer Science Department

University of Houston - Clear Lake

Houston, TX 77058

Yang.Andrew.T@acm.org

Ralph F. Grove

Department of Computer Science

James Madison University

Harrisonburg, VA 22807

Ralph.Grove@acm.org

Abstract

With the growing use of Web technologies in various critical applications, it is important for a Computer Science student, whether undergraduate or graduate, to gain experience in developing multi-tier Web-based applications. These applications provide access to data and service from a remote server, which may in turn access databases distributed across the enterprise network or the Internet. The fact that the Web is becoming the next-generation computing platform makes the introduction of Web-based application development into the Computer Science curriculum an important task. In the new *Computing Curricula 2001* report [6], *net-centric computing* is one of the components in the *Computer Science Body of Knowledge*, as recommended by the joint ACM/IEEE Task Force on Computing Curricula. In this paper, we first analyze the current practice of teaching Web-based Development, including our own experience, and then propose an integrated design of a *Web-based Development* course to be adopted as an *NC3* course (*Building Web Applications*) as categorized in *Computing Curricula 2001*.

Keywords: web-based development, curriculum design, computer science education, web-based information systems

Approximate word counts: 6,310

1. Introduction

With the growing use of Web technologies in various critical applications, it is important for a Computer Science student, whether undergraduate or graduate, to gain experience in developing multi-tier Web-based applications. These applications provide access to data and services from a remote server, which may in turn access databases distributed across the enterprise network or the Internet. The fact that the Web is becoming the next-generation computing platform makes the introduction of Web-based application development into the Computer Science curriculum an important task. In the new *Computing Curricula 2001* report, *Net-Centric Computing* is one of the components in the *Computer Science Body of Knowledge* , as recommended by the joint ACM/IEEE Task Force on Computing Curricula. (The *Computer Science Body of Knowledge* is attached as Appendix A of this paper.)

The *Computer Science Body of Knowledge* is composed of 14 modules. Core hours are recommended for all but two of the modules. The total number of recommended core hours is 277. The modules, as well as their respective recommended core hours, are listed in Table 1. The respective percentages of recommended core hours, when compared against the total recommended core hours, are also included in the table. The *Net-Centric Computing* module, for instance, contains 15 core hours, which is 5% of the total recommended core hours for the *Computer Science Body of Knowledge*.

Modules	Recommended core hours	Percentage of the core
PF. Programming Fundamentals	65	23%
DS. Discrete Structures	37	13%
AR. Architecture	33	12%
AL. Algorithms and Complexity	31	11%
SE. Software Engineering	30	11%
OS. Operating Systems	22	8%
SP. Social and Professional Issues	16	6%
NC. Net-Centric Computing	**15**	**5%**
IM. Information Management	10	4%
IS. Intelligent Systems	10	4%
PL. Programming Languages	5	2%
HC. Human-Computer Interaction	3	1%
CN. Computational Science	0	0%
GR. Graphics	0	0%
Total recommended core hours	**277**	**100%**

Table 1: Modules and their respective recommended core hours

It is worth noting that, among the eight recommended sub-modules in the Net-Centric Computing module (see Appendix A), the first two sub-modules (*NC1 Introduction to net-centric computing* and *NC2 The web as an example of client-server computing*) are recommended to be part of the core CS curricula. Although 5% does not seem to be a big percentage, the inclusion of the two sub-modules in the core CS curricula somehow indicates the increasing importance of net-centric computing and, in particular, Web-based computing in CS curricula.

In this paper, we first analyze the current practice of teaching Web-based Development, including our own experience, and then propose an integrated design of a Web-based Development course to be adopted as an NC3 course (*Building Web Applications*) as categorized in the new Computing Curricula 2001. Although NC3 is not part of the core, it is apparently an important topic, which we expect will be offered by more colleges and universities as an upper level elective.

2. Current Practice in Teaching Web-Based Development

Curriculum

At present many universities offer beginning-level Web-related courses focusing on a single technology, with titles such as *Java programming* , *Internet programming with Java*, *Web programming using ASP* , et. al. We believe more schools will start to offer advanced Web development courses, especially when Computing Curricula 2001 is formally released by the IEEE/ACM Joint Committee. A few efforts to develop more comprehensive courses in web-based application development have been reported at conferences [1] [2] or published on the Internet [3]. Some commonalities of experience emerge from these reports, though the courses differ somewhat in objective and program context.

- Teaching web-based development is like trying to hit a moving target. The technologies used to develop web applications are constantly changing. Not only are new tools being developed continuously, but the existing tools are being upgraded, enhanced, and modified. As a result, a web-based development course must be revised continuously and the course itself becomes obsolete quickly.

- There is a great deal of applicable technology. It is not possible to include all web technology in a particular course, nor is it desirable, since students will be faced with possibly a completely new set of tools by the time they enter industry. Our best bet, as always, is to stress design principles that underlie the technology, which can be applied with newer tools as well.

- A web-based development course must fit its curriculum. Whether students study C or Java, whether they study user interface or network engineering more or less, and whether they study data base development as part of the course ultimately depends upon the mission of the organization in which the course is being taught and upon how the course is complemented by other existing courses.

Textbooks

It is a common understanding that a college level Computer Science course, especially one at the upper level, should integrate the theoretical knowledge with practical examples illustrating the application of the knowledge. Take an upper level course on *Fundamentals of Programming Languages* as an example. The course usually starts with a classification of programming languages (procedural, functional, logic, object-oriented, et. al.), followed by discussion of language features with examples of specific programming languages from each of the categories. It is hard to imagine that an instructor of such a course would use one book covering only a specific language, whatever it is, because a single language would not provide the comparative perspective that students are supposed to acquire from such a course. It is equally hard to imagine that the instructor would use multiple books, one for each type of programming languages, because students would not be able to afford the cost and, even worse, the combination of those books still may not provide an organized coverage of the fundamental knowledge underlying programming languages. (The sum of parts is not equal to the whole!)

While most instructors teaching an upper-level course on *Fundamentals of Programming Languages* would pick the textbook by Pratt & Zelkowitz [4] , or one by Sebasta [5] , or any book similar to those two, there exist no appropriate textbooks for upper-level courses on Web-based Development. Although there exist an abundance of books on Web technologies, most of them are not appropriate for being adopted as the primary textbook in an upper-level Computer Science course. All of those books that we have reviewed suffer at least one of the following drawbacks:

1. They cover the technology but not sufficient details about the fundamental knowledge underlying the technology. That is, they are better classified as reference books or technical manuals rather than as textbooks.

2. They focus on a single technology (Java, ASP, PHP, CGI, et. al.) and do not provide the readers a comparative perspective of the technologies, which is essential in college education. Most of those books may be appropriate for courses taught in technical schools or community colleges, but not for advanced courses in a four-year college or graduate schools.

The authors have both recently taught Web-based development courses and have not found any books with the desirable features, that is, a book that covers the fundamentals, the architecture & design, and at least two major technologies. All the

books we have reviewed focus on a single technology, and usually do not have enough material on topics covering the fundamentals, the architecture and the design. As examples, some of the books addressing Web development are listed below:

Francis et. al., Beginning Active Server Pages 2.0, Wrox, 1998.

Chase, Active Server Pages 3.0 from Scratch, QUE, 2000.

Welling and Thomson, PHP and MySQL Web Development, Sams, 2001.

Wall et. al., Programming Perl (3rd edition), O'Reilly & Associates, 2000.

Hall, Core Servlets and Java Server Pages, Prentice Hall, 2000.

Flanagan, Javascript: The Definitive Guide (3rd edition), O'Reilly & Associates, 1998.

Leinecker, COM+ & XML: ASP.NET on the Edge, Hungry Minds, Inc., 2001.

McLaughlin & Loukides, Java and XML, O'Reilly & Associates, 2000.

Tittel, et. al., HTML 4 for Dummies, Hungry Minds, Inc., 2000.

Hall and Brown, Core Web Programming, Prentice Hall, 2001.

The lack of appropriate textbooks for teaching Web-based development was reported as early as 1998 [1], and it is quite unfortunate that the problem continues to exist as of today. As the majority of the Computer Science educators would agree, the ideal textbook for a course on *Fundamentals of Programming Languages* in the college setting is one with balanced treatment between the fundamental concepts and the languages, and covers the major representative programming languages. A similar argument can be made concerning the design of a college-level Web-based development course. That is, such a course shall not cover only a specific technology. Rather it shall provide balanced coverage of fundamental knowledge and technologies, while employing more than one Web technology as illustration of the fundamental concepts.

As a matter of fact, the technologies and methodologies in the area of Web-based development are still changing rapidly. It is not difficult to find, in a publisher's catalog or website, sub-categories such as 'programming languages/comparative languages' or 'theory of programming languages' under the 'Programming Languages' category. What you are most likely to find under the 'Web Programming and Design' category, if it exists at all, are listing of individual technologies. To illustrate this phenomenon, a sample listing of categories from the website of a major Computer Science publisher is included as Appendix B . When the knowledge of this field becomes more mature, we expect to see sub-categories such as 'Web Development Languages / Comparative Languages' or 'Web-based software design' to be added to publishers' catalogs, and more courses such as 'Web-based Development' to be added to Computer Science curricula throughout the world.

3. Designing and Teaching a Web Development Course - an Experience Report

Teaching a Web Development Course

A new course titled *COSC415 Internet Architecture & Programming* was designed and offered to Computer Science students in the semester of Fall 2000 at IUP. Included in the course were two major server-side technologies, Java servlets and ASP (Active Server Pages), as well as introduction to fundamental concepts and other techniques, including HTML, client-side scripting (JavaScripts, VBScripts), JSP (Java Server Pages), and RMI (Remote Method Invocation), etc.

At the time the course was first offered (Fall 2000), the prerequisites included an object-oriented programming course (either Java or Visual Basic) plus some exposure to SQL syntax, which could be satisfied by taking a database course either before or at the same time. The prerequisites, however, were not strictly enforced. Among the 12 enrolled students, four of them did not have either Java or Visual Basic, while the other students had either Java or VB or both. Table 2 shows the background of the students and the respective grades that they earned.

Previous experience	Number of students in each category	Final grades
Both VB and Java	1	A
Java but no VB	5	AABCC
VB but no Java	2	BC
Neither Java nor VB	4	BCWW

Table 2: Categorization of students based on their background

The average grade with respect to each of the categories is summarized in Figure 1. Grade A is equivalent to 4, B to 3, and so on, and grade W (Withdrawal) is equivalent to zero. It was no surprise that students with both Java and VB did much better than the other students, and students without either of the two languages did not do as well as the others. What we found interesting was that students with Java but no VB experience performed better than those with VB but no Java experience.

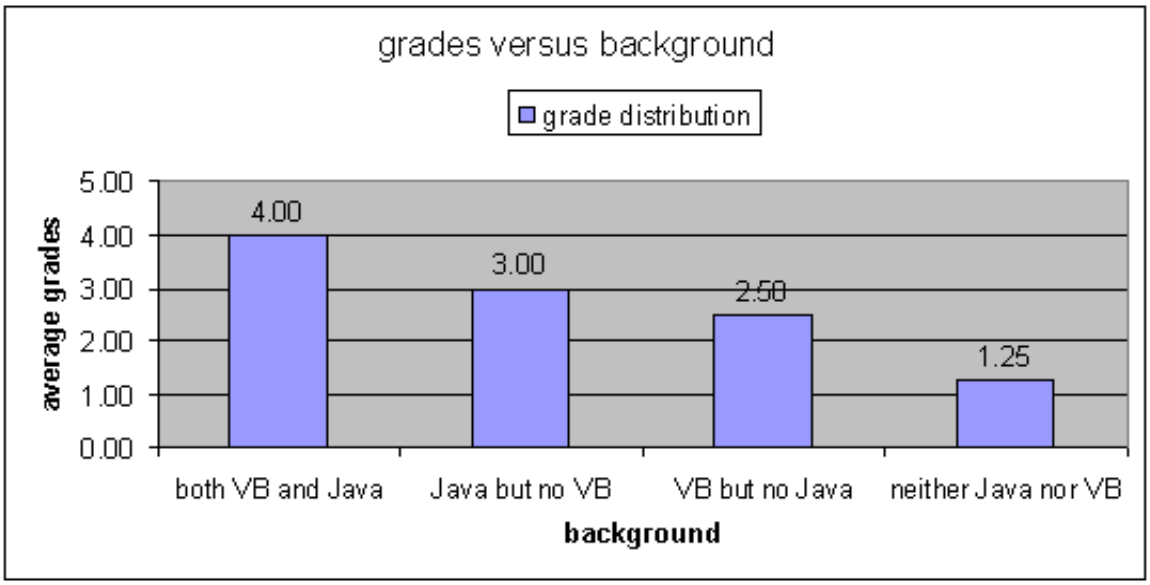

Figure 1: Grades versus student background in the first offering

The rationale underlying the decision of the original pre-requisites, which turned out to be incorrect, was that, since both Java and VBScripts were to be used in that course, it should suffice if a student had either Java or Visual Basic as part of his or her background, and their experience with either of the object-oriented programming languages should allow them to pick up the other language with some reasonable effort. Very early in the course, around the time of Assignment 2 when students were required to use both Java and VBScripts, it was realized that the hypothesis was invalid. While students with Java but no VB experience had no problem picking up VBScripts, students with only VB experience found it difficult to learn sufficient Java in order to complete the programming assignments. This observation eventually led to the change of having *Java* and *Data Structure* as the prerequisite, while *Database Systems* as the co-requisite.

After a series of revisions, especially upon the pre-requisites, the final version of the design of that course was completed in December 2000, and is available on line at http://www.cosc.iup.edu/programs/syllabi/415syl.html .

Lessons Learned from the First Offering

Through the instructor's observation and discussions with the class, a few good lessons were learned out of teaching *COSC415 Internet Architecture & Programming*.

- Student background: A good grab of Java is a must! Some students who took a Java course before still had difficulty successfully completing the programming assignments. More coverage of fundamental topics in Java, such as Java servlets, will be helpful, especially during the first quarter of the course. An alternative approach is for the Java programming course, which is one of the pre-requisites, to cover more depth, by including Java servlets as one of its modules.
- Comparative technologies: ASP can be easily learned even for students without Visual Basic experience. On the other hand, students with only Visual Basic experience found it challenging to learn Java and completing the programming assignments that require the use of Java.
- Fundamental concepts: It is important to highlight the multi-tier model and the distinction between client side and server side development, at the very beginning of the course. A reasonable understanding of the underlying distributed model is critical for the learning of later topics, especially how the various technologies such as Java servlets, JSP, ASP, JDBC, ADO, et. al., would be positioned in the distributed environment.
- Textbooks: Teaching the *Internet Architecture and Programming* course was challenging, partially due to lack of suitable textbooks that would provide comprehensive coverage of the architecture, the protocols, and the various technologies. As discussed earlier, most books focus on a single technology and lack the comprehensive coverage that is critical in an advanced college-level course.
- System support issues: Another major challenge in teaching the course is the fact that multiple technologies, including Java servlets, JSP, ASP, JDBC, ADO, RMI, et. al., need to be covered in the course and, hence, software supporting those technologies need to be installed in a distributed environment. The instructor found it a constant need to install, test, re-install, and re-test various software and APIs in order to use them in the course.
- Curricular design alternatives: Through a detailed discussion with the class, it was determined that a new course on *Distributed Component-Based Development*, covering component-based software development using technologies such as CORBA, EJB, RMI, and DCOM, is highly desirable. Students who have completed the *Internet Architecture & Programming* course may take the *Distributed Component-Based Development* course to further their knowledge in the domain of distributed software development. Both of the two courses use multiple languages, such as Java and VBScript, to illustrate the underlying architecture.

Instead of having two courses, one on Web-based development and the other on distributed components-based development, an alternative design is to divide the two courses by languages: one course focusing on Java while the other focusing on VBScript. Both courses would cover Web-based as well as components-based development. The Java based course, for example, would cover Java servlets, JSP, JDBC, Java Beans, EJB, RMI, and CORBA. The VB based course, on the other hand, would cover ASP, ADO, COM, DCOM, and CORBA.

After discussion and debates, the class voted for the 1st approach (~84%). The main argument leading to the "victory" of the 1st approach was that, with the 1 st approach, while most students may not take both the Web-based development and the distributed components-based development courses, taking one of them would enable the students to have broad perspectives on the comparative technologies, such as Java and VBScripts. With the 2nd approach, students who take only the VB course, for example, may not develop the appreciation of other languages for Web and distributed development. The 1st approach divides the two courses by the underlying architecture, while the 2 nd approach makes the division based on the implementation languages and tools.

4. Design of an NC3 Course

A Web-Based Development course might be implemented in any of a variety of formats depending on the degree of significance attached to the topic and the mission of the program in which it is taught. The course could be packaged for a single semester or as a series of two courses, or for an equivalent number of quarters. It might also incorporate an emphasis on software engineering, e-commerce, networking, artificial intelligence, or another related topic. More or less emphasis might be placed upon certain aspects of building web-based applications, such as GUI development, communication protocols, and database design. The course described here is a one-semester course, of three credit hours, that focuses on providing students an understanding of and experience with multi-tier web-based applications.

Relatively more emphasis is placed within the course upon the middle layer of the multi-tier architecture, the business logic service, since this part of the multi-tier application has less coverage in the traditional curriculum than either front-end design or database services. The focus is on server-side programming of application logic, with enough of the outer layers included as is necessary to complete students' understanding of the issues, technology, and design patterns prevalent in web-based applications. The course is thus complementary to existing curriculum that covers interfaces and database design.

One important question that arises in teaching the course is what set of technologies to use. Server-side component design can be based in Java, with servlets, JSP, JDBC, and related API's, or it might be based in CGI scripts, or it might comprise ASP and related API's. Rather than provoke a language-war, our approach is to include an overview of available technologies to students and to then select the technology set that is most practical, based upon the instructor's experience, students' background, and the available tools and platforms for development. Thus, students might develop Java servlets one semester, and use ASP the next. The central design themes and architectural patterns remain constant, however. As we learned from the first offering of *COSC415* (see section 3), a "favorable" alternative is to teach Web-based development in the first semester, using multiple languages, while teaching components-based development in the next semester, also using multiple languages.

Course Objectives

We believe that course design should always begin with a clear statement of student-centered learning objectives. Our objectives are stated in demonstrable terms, to give students a clearer idea of what is expected of them, and to use as a basis for creating exams.

Upon completion of the course, students should be able to:

- Explain the use and general format of principal Internet communication protocols
- Describe the purpose, strengths, and weaknesses of client-side program components, including browsers, scripts, and applets
- Describe the purpose, strengths, and weaknesses of server-side program components, including CGI scripts, ASP, JSP, PHP, and servlets
- Explain the structure of three-tier and N-tier web-based applications and describe the required software components
- Develop small web-based applications including an HTML interface, a database interface, and a server application to provide complete system functionality
- Effectively employ security protocols in developing web-based applications
- Use appropriate methodology for design, development, and testing of web-based applications

The first four objectives are of a general nature, related to principles of developing web-based applications. The last three are more functional, relating to actual development of web-based applications, which is the culmination of the course curriculum.

Prerequisites

Prerequisite knowledge and ability is kept to a minimum, in order to make the course open to as wide an audience as possible. Since a significant part of the course is practical, involving application development, programming skills and application development experience is a must. The application development experience need not be extensive, however. A second course in programming (even if the first was at the high school level) is generally adequate. It would be best, of course, for students to have had experience using the particular programming language that is being used in the course (Java, Perl, etc.). At least a closely related language is necessary. It is easy for C++ programmers to learn to write Java code, but much more difficult for Visual Basic programmers to do so, for example.

Course Content

The course is divided, chronologically, into four sections. The first two sections provide an introduction and overview of the technology and design issues involved in building web applications. The third section supports the development of a web-based application. The final section includes a slightly expanded view of the web-based application context.

The goal of section one is to give students a general introduction to relevant protocols, languages, and technologies. It is, in a sense, an introduction to the “language” of web development. At the same time, this section is intended to help students develop a conceptual framework for understanding the global design issues that are to follow.

Section I: Fundamentals

a. Networking Fundamentals: sockets, IP, TCP, HTTP
b. Client-side Components: HTML, XML, browsers, applets, HTML forms, JavaScript, VBScript.
c. Server-side Components: Web servers, servlets, CGI, JSP, ASP, PHP, application servers.
d. Data-base Components: SQL, JDBC, database servers

Section two develops a more holistic view of web-based applications by focusing on issues of design and architecture. Typical web-based application architectures are reviewed, from simple client-server interaction to complex multi-tier applications with distributed services. Different varieties of services are examined, including transaction, database, and directory and legacy servers. Security issues such as identification, confidentiality, privacy, and related technologies are included here as well, since they should be an integral part of any application design.

Section II: Architecture and Design

a. 1..N-tier designs
b. User interface versus business logic
c. Back end servers: database servers, directory servers, transaction servers, interfacing to legacy systems
d. Security issues
e. Object-oriented modeling for Web applications using UML & WAE

The third part of the course presents the technologies and programming techniques necessary to build the middle layer of a complete multi-tier application. This particular topic list includes Java-related and ASP-based technologies, but it could easily be replaced by any two functionally equivalent set of tools, at the instructor’s discretion. Our feeling is that once students understand the functionality of the middle layer of software, that knowledge can easily be transferred to any appropriate technology set.

As discussed in section 2 (Current Practice), a common "problem" of teaching Web-based development is that very often the course focuses too much on a single technology. By using at least two comparative server-side technologies, the course provides the students a broader picture of Web-based development. A "comparative" approach will also strengthen the students' understanding of the underlying concepts and Web-based architecture.

Section III: Development of multi-tier applications

Java-related

a. Setting up the Java servlet engine
b. Fundamental Java servlets: servlet life cycle, Request and Response
c. Session Management and cookies
d. Dynamic HTML generation
e. Data Management using Java Database Connectivity (JDBC)
f. Java Server Pages
g. Integrating Java servlets and JSP

ASP-based

a. ASP Object Model
b. Fundamentals of ASP: getting started
c. Basic ASP: input, output, variables, arithmetic operations, strings, arrays
d. Control structures
e. Object-Orientation in ASP: objects, properties, methods and events
f. Session Management: Applications, Sessions, Cookies
g. Database Access with ASP: ADO object model

The final section of the course is intended to give students a look at the next level of complexity in building web-based applications. Topics include component frameworks for heterogeneous distributed applications, directory access, and search engine interfacing. In addition to providing a look over the horizon for students, this section provides material for discussion in class while students complete the development of a web-based application using information learned in the previous section.

Section IV: Special Topics

a. Distributed component frameworks (COM/DCOM, CORBA, RMI, Jini)
b. Secure directory access using LDAP
c. Search engines

Pedagogy

The course design depends on three critical pedagogical components: a running case study, a significant project assignment, and teamwork.

The role of the case study is to provide an ongoing basis for examples that illustrate technologies presented in class. Having a consistent case study throughout the class provides students a familiar context in which new techniques can be explained, and it helps to illustrate how various techniques are integrated. An HTML form might be developed to illustrate the construction of a user interface, for example, and then that same form would be used to explain server-side processing of the data it contains. The case study also serves as an example for project development work done by the students. A web chat facility requiring user registration and login is a possible case study – it requires a user interface, server-side processing, and persistent data, which are the essential elements of the applications being studied.

Completing a significant project that is a web-based information system of some sort will give students the chance to integrate what they have learned and to synthesize a solution to a new problem. We know that students also derive satisfaction from building their own useful working application, especially one that has a visual (user interface) component. The scope of the project is critical, however. It must address the learning objectives of the course by incorporating the principle material covered, it must be different enough from the case study that was presented so that it requires creative thinking, but it must be manageable within the available time that students can be expected to contribute to the class. A records management application (directory, inventory, etc.) or a textual web browser (with memory) would be appropriate, for example. If possible, students should be given the opportunity to present their finished products to the class, so that they can learn from each other and so that they get the satisfaction of showing their finished work to the public.

Having students work in teams as they complete their project has several benefits. First, it makes development of a moderate-sized web-based application more practical, since the work can be shared among two or three students. This is especially important if the project product is to include design or narrative elements. Working in teams also gives students an excellent opportunity to learn from each other. Their shared experiences can be more enlightening than time spent in the classroom. One additional benefit is that assigning projects to teams means that fewer individual projects are completed, and this makes it practical for students to present their finished product in the classroom (eight presentations instead of 24, for example).

Tools and Facilities

In order to try out the techniques presented in class and to complete their assigned project, students will need access to the complete set of tools and servers that support web-based applications.

One or more web browsers must be available to test user interfaces and system functionality. This is not a big problem, since most available computers have one or more browsers available. We recommend that students test all code that they develop

using both Netscape and Internet Explorer at least. For thorough testing, students can also test using multiple versions of these, and with other less widely used browsers (e.g. Opera, Mozilla, and Konquerer).

Though the use of integrated development environments sometimes generates controversy, we advise students to use such tools for development of HTML as well as server-side code. Once students understand the essential syntax of a language and can effectively write code, there is little to be gained from denying them the use of integrated tools. There are a variety of freeware and commercial tools available, so finding an appropriate tool set is usually not a problem.

As they develop a project, students will need certain normally restricted forms of access to servers, which can be a problem. To test CGI scripts, servlets, and active server pages, students will need permission to place their code in the appropriate server directories. Even more important, however, students will be writing code that runs at a privileged level in the operating system, which raises the possibility of accidental (possibly malicious as well) damage and improper access to system resources. The simple solution to the problem is to provide a server for the exclusive use of students in the class, so that damage will be contained and can be easily repaired. Depending on the robustness of the particular server tools being used, however, it might be necessary to frequently restart or even rebuild the server. A team of students from the class can be given responsibility for this as well, giving them additional good experience in managing web-based applications.

To complete a comprehensive project, students will need access to a database server as well. In this case, however, they need only enough information to read and write the database, which can be initialized beforehand. The same set of data can be used to populate an initial database for each project team, and procedures for re-initialization of the data can be created so that students can start with a clean data set when testing their projects.

Possible Alternative Designs for Follow-On Courses

The course described above is a one-semester comprehensive course in web-based information system development. A more extensive treatment of the topic, in two semesters, might expand the core topics in any of several different directions. One possibility would be to include more about user interface development, including both graphical design and the use of client-side scripts such as JavaScript or VBScript. Another possibility would be to include more coverage of database development and database tools. A second semester would also allow students time to look at a more diverse set of development technologies, including CGI, ASP, JSP, PHP, etc. There are several related topics that would complement the core topics as well, including cross-platform connectivity (CORBA, DCOM, RMI, Jini), and various domain-specific API’s.

5. Summary

There is a clear need for treatment of web-based development in the CS curriculum. The growing significance of the World-Wide Web as a medium for business, education, commerce, government, and virtually all human communications implies that all information technology specialists will need to be familiar with web-based applications. This fact is reflected in the inclusion of net-centric computing in Computing Curriculum 2001.

Teaching a web-based development course presents several issues to curriculum designers. One issue is choosing a set of technologies to present to students from among the many that are available. Another is deciding what relative emphasis to place upon networking protocols, server-side operation, client-side operation, and database services, which are the primary components of the multi-tier application architecture. Related issues such as information security and software engineering must be considered as well. Finally, there are practical issues, such as how to provide a flexible but secure development environment for students, and how to keep abreast of changing technology.

The course design presented here reflects the requirements of Computing Curriculum 2001 as well as the authors' practical experience in teaching a web-based development course. This design is not offered as a one-size-fits-all solution, but rather as a practical guide to developing a course that is appropriate for the environment for which it is taught.

6. Appendix A: Computer Science Body of Knowledge [6]

DS. Discrete Structures (37 core hours)
DS1. Functions, relations, and sets (6)
DS2. Basic logic (10)
DS3. Proof techniques (12)
DS4. Basics of counting (5)
DS5. Graphs and trees (4)

PF. Programming Fundamentals (65 core hours)
PF1. Algorithms and problem-solving (8)
PF2. Fundamental programming constructs (10)
PF3. Basic data structures (12)
PF4. Recursion (6)
PF5. Abstract data types (9)
PF6. Object-oriented programming (10)
PF7. Event-driven and concurrent programming (4)
PF8. Using modern APIs (6)

AL. Algorithms and Complexity (31 core hours)
AL1. Basic algorithmic analysis (4)
AL2. Algorithmic strategies (6)
AL3. Fundamental computing algorithms (12)
AL4. Distributed algorithms (3)
AL5. Basic computability theory (6)
AL6. The complexity classes P and NP
AL7. Automata theory
AL8. Advanced algorithmic analysis
AL9. Cryptographic algorithms
AL10. Geometric algorithms

PL. Programming Languages (5 core hours)
PL1. History and overview of programming languages (2)
PL2. Virtual machines (1)
PL3. Introduction to language translation (2)
PL4. Language translation systems
PL5. Type systems
PL6. Models of execution control
PL7. Declaration, modularity, and storage management
PL8. Programming language semantics
PL9. Functional programming paradigms
PL10. Object-oriented programming paradigms
PL11. Language-based constructs for parallelism

AR. Architecture (33 core hours)
AR1. Digital logic and digital systems (3)
AR2. Machine level representation of data (3)
AR3. Assembly level machine organization (9)
AR4. Memory system organization (5)
AR5. I/O and communication (3)
AR6. CPU implementation (10)

OS. Operating Systems (22 core hours)
OS1. Operating system principles (2)

GR5. Visualization
GR6. Virtual reality
GR7. Computer animation
GR8. Advanced rendering
GR9. Advanced geometric modeling
GR10. Multimedia data technologies
GR11. Compression and decompression
GR12. Multimedia applications and content authoring
GR13. Multimedia servers and file systems
GR14. Networked and distributed multimedia systems

IS. Intelligent Systems (10 core hours)
IS1. Fundamental issues in intelligent systems (2)
IS2. Search and optimization methods (4)
IS3. Knowledge representation and reasoning (4)
IS4. Learning
IS5. Agents
IS6. Computer vision
IS7. Natural language processing
IS8. Pattern recognition
IS9. Advanced machine learning
IS10. Robotics
IS11. Knowledge-based systems
IS12. Neural networks
IS13. Genetic algorithms

IM. Information Management (10 core hours)
IM1. Database systems (2)
IM2. Data modeling and the relational model (8)
IM3. Database query languages
IM4. Relational database design
IM5. Transaction processing
IM6. Distributed databases
IM7. Advanced relational database design
IM8. Physical database design

NC. Net-Centric Computing (15 core hours)
NC1. Introduction to net-centric computing (9)
NC2. The web as an example of client-server computing (6)
NC3. Building web applications
NC4. Communication and networking
NC5. Distributed object systems
NC6. Collaboration technology and groupware
NC7. Distributed operating systems
NC8. Distributed systems

SE. Software Engineering (30 core hours)
SE1. Software processes and metrics (6)
SE2. Software requirements and specifications (6)
SE3. Software design and implementation (6)
SE4. Verification and validation (6)
SE5. Software tools and environments (3)

OS2. Concurrency (6)
OS3. Scheduling and dispatch (3)
OS4. Virtual memory (3)
OS5. Device management (2)
OS6. Security and protection (3)
OS7. File systems and naming (3)
OS8. Real-time systems

HC. Human-Computer Interaction (3 core hours)
HC1. Principles of HCI (3)
HC2. Modeling the user
HC3. Interaction
HC4. Window management system design
HC5. Help systems
HC6. Evaluation techniques
HC7. Computer-supported collaborative work

GR. Graphics (no core hours)
GR1. Graphic systems
GR2. Fundamental techniques in graphics
GR3. Basic rendering
GR4. Basic geometric modeling

SE6. Software project methodologies (3)

CN. Computational Science (no core hours)
CN1. Numerical analysis
CN2. Scientific visualization
CN3. Architecture for scientific computing
CN4. Programming for parallel architectures
CN5. Applications

SP. Social and Professional Issues (16 core hours)
SP1. History of computing (1)
SP2. Social context of computing (2)
SP3. Methods and tools of analysis (2)
SP4. Professional and ethical responsibilities (2)
SP5. Risks and liabilities of safety-critical systems (2)
SP6. Intellectual property (3)
SP7. Privacy and civil liberties (2)
SP8. Social implications of the Internet (2)
SP9. Computer crime
SP10. Economic issues in computing
SP11. Philosophical foundations of ethics

7. Appendix B: Sample Listing of Computer Science Areas [7]

Programming Languages and Theory

- Programming Languages/Comparative Languages
- Theory of Programming Languages
- Logic Programming
- Functional Programming
- Natural Languages

Software Engineering (SE)

- Software Engineering (SE)
- Software Engineering--Advanced
- Software Testing and Verification
- Software Reliability
- Software Requirements
- Programming Methodology
- Performance Modelling
- Computer-Aided Software Engineering
- Software Maintenance
- Software Metrics

Web Programming and Design/E-Commerce

- Web Programming and Design

- Active Server Pages (ASP)
- CGI Programming
- HTML
- HTML: Advanced
- Javascript/Java for Web Programming
- PERL--Programming
- PHP
- SGML
- XML
- Electronic Commerce: Programming and Technology
- Electronic Commerce: Business Issues
- Software Applications for Web Authoring, Design, and Management
- Web Design
- Flash
- FrontPage

8. References

[1] Lim, Billy B.L., “Teaching Web Development Technologies in CS/IS Curricula”, Proceedings of SIGCSE 1998, Atlanta, GA, 1998, pp. 107-111.

[2] Klassner, Frank. “Can Web Development Courses Avoid Obsolescence?”, Proceedings of ITiCSE 2000, Helsinki, Finland,, pp. 77-80.

[3] Greenspun, Phillip. “Software Engineering for Internet Applications”, http://philip.greenspun.com/teaching/one-term-web , accessed November, 2001.

[4] Terrence W. Pratt, Marvin V. Zelkowitz, *Programming Languages: Design and Implementation, 4th edition*, Prentice Hall, 2001.

[5] Robert W. Sebesta**,** *Concepts of Programming Languages* , Addison-Wesley, 1998.

[6] ACM/IEEE 2001: *Computing Curricula 2001*, recommended by the ACM/IEEE Joint Committee. The current version of the report is available at http://www.computer.org/education/cc2001/report/ .

[7] See http://vig.prenhall.com/catalog/academic/discipline/1,4094,591,00.html