A "CS 1.5" Introduction to Web Programming

Marty Stepp
Computer Science & Engineering
University of Washington
Seattle, WA
stepp@cs.washington.edu

Jessica Miller
Computer Science & Engineering
University of Washington
Seattle, WA
jessica@cs.washington.edu

Victoria Kirst
Computer Science & Engineering
University of Washington
Seattle, WA
vkirst@cs.washington.edu

ABSTRACT
Web programming is increasing rapidly in importance at the university level, yet there is no consensus about when and how it should be incorporated into the computer science curriculum. This paper describes our results in teaching an experimental introductory web programming course at the University of Washington that has had great success in attracting large numbers of students from inside and outside the computer science major. The course requires CS1 as a prerequisite, striking a good balance between making the course open to non-majors but also more rigorous for students with programming background. We classify the course as "CS 1.5" because many of our students take it between CS1 and CS2. We use our evaluation data to argue that a web programming course at this level leads to a great deal of student interest and enthusiasm, broadens the reach of computer science, and provides a valuable service to other departments.

Categories and Subject Descriptors
D.1.0 [**Programming Techniques**]: General—*Web*.
H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces --- Web-based interaction.
K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*.

General Terms
Human Factors, Languages, Measurement, Standardization.

Keywords
Pedagogy, Computer Science Education, Web Programming, CS1.5, HTTP, HTML, XHTML, CSS, PHP, JavaScript, XML, Ajax, SQL, databases, web security.

1. INTRODUCTION
More and more of the world's software is being run within a web browser. Web software offers many legitimate benefits: ease of deployment, ubiquity of access to a global audience, and availability of server-side data and services. More recently, more mature web standards and technologies such as Ajax have taken the web beyond simple document processing to the current "Web 2.0." With these advances the relevance of teaching web programming at the college level has also increased.

But web programming's role in the computer science curriculum has not yet been clearly defined. Many universities teach web programming only as part of information science degrees or other programs separate from computer science. This may reflect the fact that some consider web programming a lesser art form, not worthy of inclusion in a comp-sci degree program. Nonetheless many CS departments possess the quirk of not teaching web programming yet offering senior-level courses that require students to complete web projects, such as software engineering, databases, HCI, and capstone project courses.

Things are slowly changing, as more computer science programs now offer rigorous web programming classes. But there does not seem to be a consensus about where in the curriculum, and at what level of detail, to introduce this material. Some offer it primarily to non-majors in CS0 at a low level of depth, focusing on HTML and JavaScript [5]. Others offer junior- and senior-level capstone or web project courses [10], often with multi-week assignments done in larger groups. The courses vary widely in the particular languages and technologies taught, particularly server-side web languages such as PHP and Ruby on Rails.

The authors led a SIGCSE 2008 Birds of a Feather session [9] in which we conversed with and informally surveyed instructors about their web courses. As shown in the following chart, essentially all attendees teach a course that covers HTML, Cascading Style Sheets (CSS), JavaScript for client-side interaction, and a server-side language. PHP was the most widely taught server language, followed by Java/JSP solutions and then others such as Ruby, Perl, and Microsoft's .NET Framework.

Table 1. SIGCSE 2008 BoF Web Language Survey Data

Language	Count
PHP	9
Java/JSP	7
Perl/CGI	4
Ruby on Rails	3
Microsoft .NET	2
Python/other	2

The BoF attendees reported that some of the variation in web programming courses is because many departments do not devote significant resources to teaching it, leaving the work to a single devoted faculty member. Others pointed out that the modern CS

SIGCSE'09, March 3-7, 2009, Chattanooga, Tennessee, USA.

curriculum is already hefty and that it is difficult to add another course to it without drawing ire from students and faculty alike.

In this paper we discuss our own results at the University of Washington with adding an introductory web programming course to our Computer Science & Engineering curriculum. During these course offerings we gathered extensive survey data from the students for assessment purposes. We will present survey data supporting our hypothesis that offering a web course early in the curriculum, particularly just after CS1, proves to be beneficial and enjoyable for students. Additionally, we will argue that offering a web course at this level also provides a valuable service to non-majors and other departments and increases enrollment in later courses in the computer science curriculum.

2. RELATED WORK

Other universities offer similar web programming courses and have published findings. Yue and Ding of Houston-Clear Lake [10], Noonan of William and Mary [4] and Jackson of Duquesne University [3] are among several who offer a senior-level course in web application development, surveying client- and server-side technologies along the way. Dealing only with CS majors at the senior level affords them the opportunity to cover technologies at a deeper level in a shorter amount of time.

Dave Reed of Creighton University [5] published his version of a non-majors CS0 focusing on basic JavaScript programming. This approach has been discussed by Zimmerman [11] and others. This model has become arguably the canonical CS0 course and is successful at many institutions. A fundamental difference from our own work is that their material is presented at a lighter level with no programming prerequisite, and that the course is largely confined to client-side programming in JavaScript without exploring other technologies. Many universities offer courses in web page design using HTML but do not focus on programming or interactive sites (many are offered by departments outside CS) and are therefore excluded from the discussion.

Michael Gousie of Wheaton College [2] offers one of the more similar courses to our own, targeting non-majors. The Wheaton course focuses on web graphics using Java applets and the AWT framework, rather than rich internet application development. Our investigations have not led us to the discovery of other published web programming courses targeting this specific audience at this level in the undergraduate curriculum.

3. OUR WEB PROGRAMMING COURSE

In September 2006, a group of educators was flown to Google for a meeting to discuss web programming in the undergraduate curriculum. Mark Lucovsky, a senior engineering manager at Google, gave a presentation encouraging educators to teach introductory web programming. Google reports to UW and other institutions that students are under-prepared to work on web software development as they complete their undergraduate studies. This is consistent with other feedback our department receives at its yearly industry affiliates' meetings, where companies ask for students more familiar with web programming. To this end, Google funded a proposal from UW to offer a web programming course once per year in 2007 and 2008.

Google's original desire was for us to convert our Java CS1 course into a web programming course, but we instead chose the lower-risk option of offering web programming as an elective course targeted at non-majors who have just completed CS1. In the common parlance this would be considered a "CS 1.5" course. Basic programming skills (loops, selection, variables, arrays, functions) are required, but no web programming experience is expected. This prerequisite proved very important, because assuming a modicum of programming knowledge allowed us to cover topics at a more brisk pace, with less focus on basic syntax and concepts like parameters and variables, and introduce more elaborate and interesting programming assignments.

Figure 1 summarizes declared majors of students enrolled in the web course in Spring 2008. Students listed as "Pre-Major" have declared a desire to entire CSE or another engineering major but have not yet applied for admission to the program.

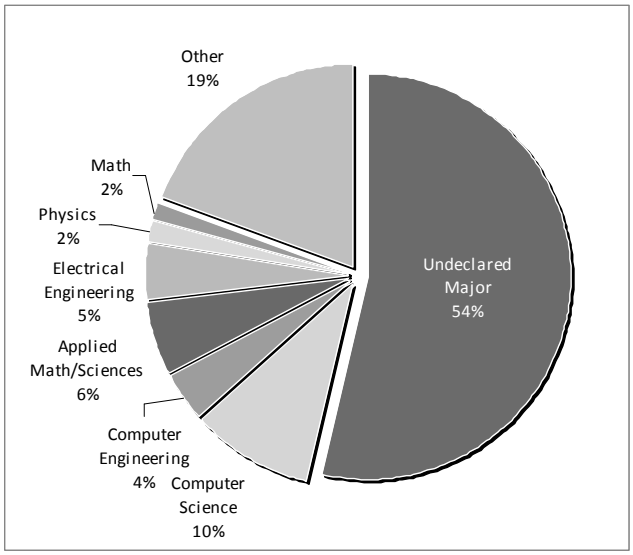

Figure 1. Student Majors in Web Course

Despite limited advertising and the course not counting toward our major, 92 students took it in 2007 and 192 in Spring 2008. 8% of the students were CS majors, and 92% were non-majors. 81% of the students were men and 19% were women.

Table 2. Web Course Topics and Assignments

Topic	Assignment
basic web pages with HTML/CSS	Granny's Pies Page
web page layout with CSS	IMDb Movie Review
JavaScript event-driven programming	ASCIImation
JS Document Object Model (DOM)	Fifteen Puzzle
Asynchronous JavaScript/XML (Ajax)	Baby Name Surfer
PHP server-side programming	To-Do List
HTML forms and server-side data	NerdLuv dating site
databases and SQL	Kevin Bacon problem

The major topics covered in the course and their corresponding homework assignments are shown in Table 2. Other topics covered in lecture include web security basics, multimedia web content, web design and usability, Google's Ajax web APIs, taking a web site "live" and managing a web server. We chose to cover a breadth of topics, achieving depth through repetition. For example, though only the first few assignments

were focused on HTML and CSS, later assignments also required and evaluated the student's knowledge of those technologies.

In our planning we decided on several major goals of the course. Unlike most other courses, we chose to teach only standards-compliant constructs and code, using entirely free software on the client and server. We focused on the most modern versions of tools and languages, ignoring compatibility with legacy software such as Internet Explorer 5.5 and HTML 4.01. We also worked constantly to keep the course's pace and difficulty at a level palatable for non-majors who have just finished CS1, reminding ourselves that most of the core audience in the class was not bound to become part of our degree program.

Another difficult decision was the choice of a server-side programming language. While the set of client-side technologies is fairly standardized (HTML, CSS, JavaScript, etc.), server-side languages vary widely between courses and textbooks. We decided to focus on PHP as our single server-side language. PHP is a flawed language, but it is simple to set up, both for the server administrator (who merely needs to install PHP onto the web server) and for the students (who can immediately upload .php files that will run). JSP, Ruby, .NET, and other languages are comparatively much more difficult to deploy and use to write small, simple introductory programs. PHP also integrates very well with Apache web server software, is completely free of charge and free to distribute, is open-source, is the most widely used and popular web language, and looks much like HTML, C, and other languages with which the students are familiar.

We chose to leave out a few important topics like web services and Flash, despite student interest in these technologies. We felt that there were enough languages and tools already in the course, and that these topics are more appropriate for a second course in web application development. It is easy for a new student to be overwhelmed by the amount of new syntax, APIs, languages, and tools that are used in a course such as this one. This is consistent with student feedback from our surveys.

The course had three 50-minute lectures per week, and one 50-minute lab session in which students would solve problems by computer with TAs available to answer questions. Students were not required to finish any particular number of problems, so long as they worked for the entire 50-minute period. Student feedback suggests that the lab sessions were very helpful; participation in the first 7 of 10 lab sessions was mandatory, but 148 out of 192 students (77%) still chose to attend the final lab session despite it not counting for course credit. Lab sessions also provided fertile ground for the instructor and TAs to discover failures in our own teaching by observing students' questions and struggles firsthand.

We believe that labs are particularly useful for this course. When programming with web languages the concepts seem easy to students, but details and bugs can be hard to find; many student bugs don't show any output and are difficult to debug. Having a lab TA to help find and fix these bugs was a crucial benefit.

Perhaps the greatest resource we have utilized in this course is our teaching assistants. We follow the model proposed by Reges [6] and also documented by Roberts [7] and Decker [1], using undergraduates to staff our labs and provide office hours to answer students' homework questions. The TAs are invaluable in helping students fix software issues, teaching them to properly debug the confusing new errors they encounter when programming for the web, and closing gaps in the instructor's explanation of the material. Good TA support is crucial when expecting non-CS majors to solve tricky web problems.

The recommended software for the course was a plain text editor such as TextPad or TextMate (Mac), along with the Firefox browser. Other tools such as the extremely valuable Firebug debugging plugin and the JsLint JavaScript syntax checker were also introduced. Students uploaded their code to a central dedicated LAMP server for the course, with each student having a private directory for storing and testing his/her programs.

3.1 CHALLENGES AND DISCOVERIES

Web programming offers many unique challenges and difficulties that we discovered while teaching this course. Probably the most prominent difficulty is the set of new nasty bugs that arise in web programming. Many of the most common student mistakes, such as misspelling a tag in HTML, forgetting a token in JavaScript, or capitalizing a variable's name incorrectly, produce no output in the browser. The student is left with no indicator that something is wrong and no clear way to find or fix the problem. There are few tools for debugging, and many of the tools that do exist are not made for new programmers. We expose students to Firebug plugin early in the course for debugging. It not only provides verbose error feedback but also the ability to dynamically change a page on the fly and see the results, as well as a full-featured JavaScript debugger and interpreter. Firebug is an excellent tool and we use it extensively in our course. We also use the W3C XHTML and CSS validators for finding mistakes in syntax in those languages, as well as the JsLint JavaScript syntax checker, which points out many common JavaScript bugs that would otherwise produce no error or warning.

Perhaps the most frustrating tool is the browser itself; incompatibility issues between browsers (largely flaws found in Microsoft Internet Explorer) make robust web programming unnecessarily difficult. We constrain our students to Firefox and instruct them to code to published web standards, ignoring quirks that may exist in Internet Explorer or other browsers. We provide links to IE-only bugs and fixes for interested students.

There is a lack of good resources from which to learn introductory web programming, particularly at the “CS 1.5” level. There are many web tutorials, but the vast majority are sloppy, out of date, or plain wrong. Sebesta [8] and Jackson [3] were the most helpful textbooks we found, but neither was a good fit for our course as both target a more advanced, 300-level web programming course. We chose not to require a textbook in Spring 2008 and instead relied on instructor-provided materials. We have written a large number of lecture notes, tutorials, and chapters for students to read about each week's material, as well as providing links to our favorite external resources and references.

Some of the challenges we faced were unique to teaching this subject and not to web programming itself. For example, the student must learn many programming languages and technologies in a short time, each introducing new syntax and new programming paradigms such as event-driven programming, client-server interaction, and so on. This is particularly tough at universities like ours that are on the quarter system. We provide "cheat sheets" and allow open-book exams to ease the pain of learning so many new languages and so much syntax so quickly.

There is a proliferation of sloppy web code examples on the various popular online tutorials, showing poor style, no comments, and otherwise poor solutions. To combat these poor examples, we rigidly enforce a two-part grading system covering "external correctness" (the program's behavior, output, and appearance) and "internal correctness" (the code's style, design, documentation, elegance, and conformance to W3C web standards), giving roughly equal grading weight to each.

A final and more subtle difficulty in a web programming course is stopping students from copying each others' work, even when said work is being placed onto the public web. To minimize code theft, we allow students to post their work only to password-protected web folders under the supervision of the course staff.

4. ASSESSMENT

During each course offering we gathered data from students. A voluntary, anonymous survey was given to each student upon submission of each assignment to assess what concepts were challenging. Mid-quarter and end-of-quarter evaluations were given to gauge satisfaction with the course and approach.

We acknowledge that self-reported data has inherent flaws. That said, the original purpose of the surveys was to gather feedback from the students to evaluate and improve the course. We share this data to begin to provide evidence that such an experimental course is worth developing. As the course matures, we hope to refine the surveys and correlate data from different terms to increase the reliability of the claims made here.

From our homework surveys, we found advanced JavaScript DOM programming and building rich Ajax Web 2.0 applications were among the most difficult concepts. This has caused us to rethink our order of topics; we plan to move basic PHP server-side programming and HTML forms earlier in the course and delay event-driven DOM programming by two weeks. We believe this will result in a more gradual difficulty curve.

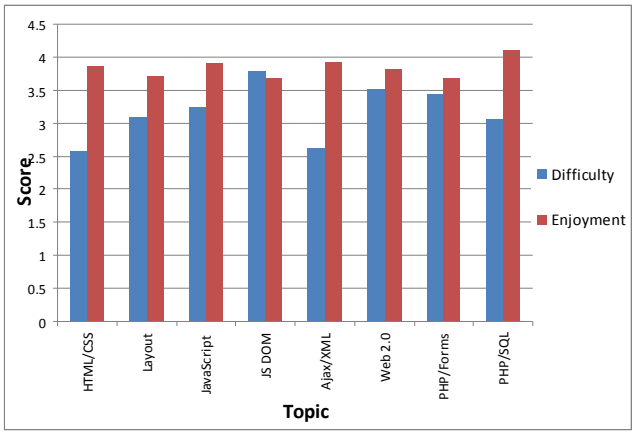

Figure 2. Homework Difficulty and Enjoyment Survey Data

Students were also asked what specific aspects of each assignment were most interesting and most frustrating. A large fraction of the students appreciated opportunities to be creative. We incorporated this into each assignment, letting the student choose background colors and images, customize text, and provide their own custom links and content as much as possible within the guidelines of the material we wanted to teach. We saw the largest impact of this on our sixth assignment, a Web 2.0 To-Do list focusing on Ajax and JavaScript effects. Students reported the most hours spent on this assignment but also one of the highest enjoyment ratings. Several students attached comments saying that they enjoyed customizing its look and behavior.

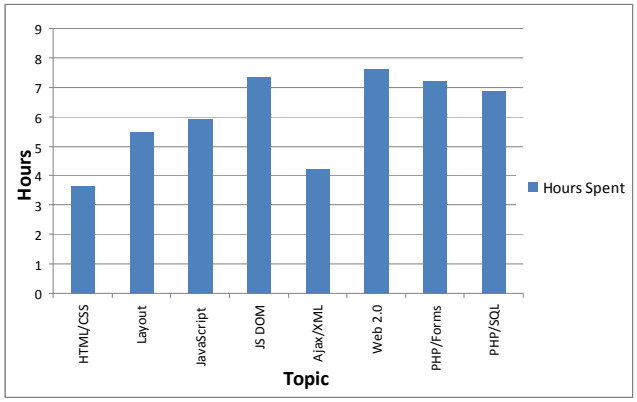

Figure 3. Homework Hours Spent Survey Data

We were encouraged to find that web programming makes it easy to incorporate student creativity into assignments. The languages involved make it simple to add pizzazz to a page without greatly increasing the difficulty or making the programs hard to grade. Students reported that they were excited that with only a few weeks of learning they could already create web sites that were near "professional quality." It is difficult to reach such a level in most early courses; this makes web programming a powerful vehicle for motivating interest in computer science.

The end-of-quarter course evaluation data shows that the course was very popular. High course ratings alone are not always compelling, but we note two particular numbers. First is the course's Challenge and Engagement Index (CEI) ranking of 6, indicating that the course was of above-average difficulty compared to other university/CS courses. The second is the low difference between average outside-of-class hours spent per week (9.0) and the number of those hours students perceived as being valuable to their education (8.6). These factors imply that though the students found the course challenging, they enjoyed it and considered almost all work in the course to be relevant and useful.

Table 3. Student Evaluation Data
(5-point scale unless otherwise specified)

Category	2007	2008
Course as a whole	4.7	5.0
Course content	4.5	5.0
Relevance and usefulness of content	4.7	4.9
Amount learned	4.7	4.9
Challenge/engagement (10-pt scale)	7	6
Hours spent on coursework per week	9.4	9.0
Valuable hours spent	7.5	8.6

5. ANALYSIS AND CONCLUSIONS

We feel that the data supports the claim that the course offering was a success. As mentioned previously, students were pleased with the course despite its high difficulty level and "work

in progress" nature. Another relevant fact is that the course currently counts for no credit toward any degree program; it counts only as general elective credit, essentially worthless to students for degree progress. That over 200 students voluntarily enrolled in such a course for no academic reward speaks to the strong student desire to see this material incorporated into the CS curriculum. Anecdotally, many students expressed a sadness that the course was ending and wished there were a follow-up course covering more advanced web concepts and larger projects. UW is in the progress of undergrad curriculum revision, and we consider such a second web course to be an area for future exploration.

A piece of feedback we received repeatedly was a sense of excitement about the course material. Simply put, students really want to learn this stuff. Web programming delivers a rich multimedia experience that brings rewarding results. The material is relevant to students, who share their work with friends, post it on Facebook and MySpace, and add it to their web sites. Web programming is interdisciplinary: Based on our data and feedback, its topics are more relevant to many non-CS majors than CS2's. Despite its reputation in some circles, web programming is conceptually deep; it gives a simple way to learn event-driven programming, to become conversant in many languages, learn the client-server paradigm, interact with databases, and more.

Having CS1 as a prerequisite freed us to cover new language syntax more quickly and therefore to solve more interesting problems. But it raises the question: Why not move web programming even later in the curriculum and write more elaborate programs? We have two counter-arguments. One is that non-majors formed the core of our large audience, and they would be unable to take the course if it had additional prerequisites. The other is that from our grade analysis, CS majors did well despite being asked to complete more difficult assignments. This suggests to us that the material is in the right place and would be too easy for junior-level students.

Another unexpected side effect of offering the web course just after CS1 is an increase in interest in CS2. This fall we see a 40-student (18%) increase in first-day CS2 enrollment, which we partially attribute to the web course's popularity. Without substantive data, our best hypothesis is that there is a large subset of students who enjoy our CS1 course but are intimidated away from taking our CS2 after hearing about its high degree of difficulty. We believe (and have heard anecdotally) that taking the web programming course gives these students another term to sharpen their programming skills and gain overall maturity as software developers before undertaking the challenge of CS2. This emboldens students who would otherwise avoid the course entirely, helping to bring additional students into our major. (This is a nice benefit of offering well-run service courses in general.)

6. FUTURE WORK

UW's web programming course will be offered in Spring 2009, again as an elective that fulfills no CS major requirements. We believe this provides some unintended benefits. It allows the course to fly in under the radar and not be subject to design and destruction by committee. It leads to a degree of self-selection, where the majority of the students in the course are there voluntarily because they want to learn the material. This promotes a positive and energetic classroom atmosphere. Also, the material and course are so exciting to students that they are willing to take it without receiving any additional reward.

We are currently collaborating with other departments such as our Informatics School to potentially allow the course to be cross-listed and to fulfill degree requirements for those departments. Cross-department interest exists for a course such as this one and helps the survival and longevity of the course.

We plan to add a weekly 50-minute TA-led discussion section to the course to match our CS1 and CS2 courses. We find that sections are immensely valuable for reinforcing the material taught in lecture. The discussions also increase TAs' feelings of ownership and investment in the course, which was a minor problem for us in past offerings. We are also developing course resources such as comprehensive lecture notes, lab exercises, discussion section handouts, video screencasts of our lectures, and a textbook. These resources will be available to other instructors. We hope to help encourage the mass adoption of introductory web programming into CS programs at this level. Our current course materials can be found at the following address:

- http://www.cs.washington.edu/190m/

7. ACKNOWLEDGMENTS

Our thanks to the SIGCSE 2007 web programming BoF attendees for their valuable insights and experiences.

8. REFERENCES

[1] Decker, A., Ventura, P., Egert, C. Through the looking glass: reflections on using undergraduate teaching assistants in CS1. SIGCSE Bulletin, 38(1): 46-50, 2006.

[2] Gousie, M. A robust web programming and graphics course for non-majors. SIGCSE Bulletin, 38(1): 72-76, 2006.

[3] Jackson, J. *Web Technologies: A Computer Science Perspective.* Prentice Hall, 2006.

[4] Noonan, R. A course in web programming. Journal of Computing Sciences in Colleges, 22(3): 23-28, 2007.

[5] Reed, D. Rethinking CS0 with JavaScript. SIGCSE Bulletin, 33(1): 100-104, 2001.

[6] Reges, S. Using undergraduates as teaching assistants at a state university. SIGCSE Bulletin, 35(1): 103-107, 2003.

[7] Roberts, E., Lilly, J., Rollins, B. Using undergraduates as teaching assistants in introductory programming courses: an update on the Stanford experience. SIGCSE Bulletin, 27(1): 48-52, 1995.

[8] Sebesta, R. *Programming the World Wide Web (4th Edition).* Addison Wesley, 2007.

[9] Stepp, M., Miller, J. Web programming in the curriculum. SIGCSE Bulletin, 40(1): 564, 2008.

[10] Yue, K., Ding, W. Design and evolution of an undergraduate course on web application development. SIGCSE Bulletin, 36(3): 22-26, 2004.

[11] Zimmermann, B. Content and laboratories of a computing science course for non-majors in the 21st century. Journal of Computing Sciences in Colleges, 19(5): 68–77, 2004.