*Behavior Research Methods, Instruments, & Computers*
*2003, 35 (2), 251-254*

# Using open-source solutions to teach computing skills for student research

DAVID W. ALLBRITTON
*DePaul University, Chicago, Illinois*

A course that relies on open-source software for teaching introductory computer programming and Web development to psychology graduate and advanced undergraduate students is described. The rationale, content, learning goals and outcomes of the course are described, along with the specific software used. The advantages of relying on open-source solutions rather than commercial software for implementing such a course are discussed.

The personal computer and the Internet have become necessary tools for research and scholarship in psychology, and almost all students develop basic competence in the use of software applications for research. Most do not, however, learn to create their own computing solutions. In this paper, I describe a course developed especially for graduate and advanced undergraduate students in psychology, Computing for Behavioral Scientists, which relies on open-source software to teach basic programming and Web development skills that can be applied to psychological research. I will describe the rationale and content of the course, the specific software used, and the advantages of relying on open-source solutions rather than commercial software for such a course.

Why teach basic programming skills to psychology students? In certain areas of psychological research (such as computational modeling), students typically develop extensive programming abilities, but most of our students could instead be described as nonprogrammers who are competent users of basic computer applications. Although students can benefit from being taught to use existing research software (see, e.g., Goolkasian, 1997), the introduction of students to computer programming in the context of research has additional potential benefits. First, there is a need for psychological software more specialized than what is available commercially, as well as for researchers to be able to develop their own (Beagley, 2001). If students understand the basics of programming, they may have better insight into the limitations of existing software, and they may contribute to the development of better research software in the future. Second, basic programming skills can give students more options for accomplishing their research objectives. Third, even when a researcher chooses to rely on available software packages or professional programmers for research software, a basic working knowledge of programming and Web development can make it easier to evaluate and choose among possible solutions. Finally, coding their own solutions to a problem can allow students to develop general problem-solving skills and confidence in their ability to learn new computer-related skills in the future. The educational benefits to students may, in fact, exceed the value of the particular applications they might develop.

The course is, therefore, aimed at advanced undergraduate or beginning graduate students who, though computer literate, are nonprogrammers or only novice programmers. The idea of teaching psychology students to use the Internet for research is not unique (see, e.g., Birnbaum, 2000, 2001), nor is this the first course designed to teach programming in the context of psychological research (see, e.g., Sargent, 1996). A distinguishing characteristic of the course described here, however, is the use of open-source software. With the development of the open-source model for software development, teaching programming to psychology students may be more useful than ever, because the open-source model provides a mechanism for the entire research community to benefit from the skills that students develop.

## Philosophy and Learning Goals of the Course

The two primary goals of the course are for students to develop problem-solving strategies that can be applied to a variety of computing tasks, and confidence in their abilities to learn new skills that they might need for specific projects. The learning goals for specific skills and content are secondary to these primary learning objectives.

The pedagogical strategy is to immediately reinforce students' efforts by having them create functioning Web pages and programs as soon as possible. Students are encouraged to experiment and "muck around with things" to find solutions to problems. In this way, the course empha-

The author thanks Ching-Fan Sheu for his advice on developing the course described in this paper, and for helpful comments on an earlier draft of the manuscript. He also thanks Jonathan Vaughan, Richard Lehman, and an anonymous reviewer for their valuable comments on previous versions of the paper. Correspondence concerning this article should be addressed to D. Allbritton, DePaul University, Department of Psychology, 2219 N. Kenmore Ave., Chicago, IL 60614 (e-mail: dallbrit@depaul.edu).

sizes that there are multiple paths through the problem space for most computing tasks. Students are encouraged to make use of man pages and Internet search engines to get ideas for solving particular problems, in order to get into the habit of finding answers and tools on their own rather than expecting the instructor to provide a step-by-step recipe. Weekly assignments typically provide students with exemplars of Web pages or programs that they then have to modify to fit a particular purpose, demonstrating a problem-solving strategy (identify relevant exemplars of solutions to similar problems and modify them to fit the current task) that students are also encouraged to use in their projects.

The specific technical skills taught in the course are:

1. The use of basic Unix commands.
2. Creating, debugging, and validating XHTML markup to create Web applications
3. Basic programming concepts and methods in JavaScript
4. The use of Perl and PHP for server-side program ming in Web applications
5. Basic database creation and manipulation, and databases in Web applications

### Course Structure and Learning Activities

Students create working examples for each course topic and link them to a home page on the course server. The central focus of the course is the final project, a Web-based application that each student develops, using skills taught in the course as well as skills identified and developed on the student's own.

The textbook is Dietel, Dietel, and Nieto's (2002) *Internet & World Wide Web: How to Program*, a useful reference for Web development as well as (in the chapters on JavaScript) a good introduction to basic programming concepts. Although we have used no other texts in this course, additional references that might be useful include texts on Unix/Linux (Siever, Spainhour, Hekman, & Figgins, 2000; Welsh, Dalheimer, & Kaufman, 1999), XHTML (Graham, 2000), JavaScript (Flanagan, 1998), Perl (Dietel, Dietel, Nieto, & McPhie, 2001; Schwartz & Phoenix, 2001; Wall, Christiansen, & Orwant, 2000), MySQL databases (Lane & Williams, 2002; Reese, Yarger, & King, 2002), and PHP (Lerdorf & Tatroe, 2002).

The students use Windows workstations to access the course server via Web browsers and telnet. Each weekly 3-h class session consists of several cycles of instruction and practice, with the instructor introducing a topic and providing a demonstration and students then working through one or more example problems.

Unix commands and utilities are first used to introduce the students to a modular and analytic problem-solving style. The students learn, for example, to use a sequence of Unix commands connected with pipes to count word occurrences within a text file—a technique that can be used to create word-frequency norms for a specialized domain not adequately characterized by general-purpose norms (e.g., Kučera & Francis, 1967).

XHTML and JavaScript are the core topics used to teach the students basic programming concepts and skills; these serve as a foundation for the rest of the course. The students learn to create Web pages with the vi and pico text editors, the free text-based HTML-Kit editor (www.chami.com/html-kit), or the open-source graphical Amaya editor (www.w3.org/Amaya).

JavaScript is used to teach basic programming concepts. Its relatively simple data structures and syntax make it relatively easy to learn, and being able to immediately see the results of their programming efforts on a Web page reinforces students' efforts. Class activities include creating a simple XHTML survey or experiment using JavaScript. Some server-side technologies are also introduced briefly as tools for implementing Web-based experiments, including databases and server-side scripting with Perl and PHP.

Although the wide range of topics limits each to only partial coverage, we have chosen this strategy for pedagogical reasons. We hoped that the experience of developing skills on their own would give students confidence in their ability to undertake new tasks in the future.

### Course Implementation Using Open-Source Software

A discarded Pentium desktop was reconfigured for use as the course server with RedHat Linux 7.1 (www.redhat.com) as the operating system. The operating system and all software used on the server are open-source, freely available over the Internet as both source code and (in most cases) precompiled binaries. Each student is given an account on the server that he/she can access via ftp, telnet, or ssh. Because of a security problem, the server was later reinstalled and upgraded to RedHat 7.2, and access via telnet and ftp was restricted to campus locations. The Apache Web server (Apache 1.3.22, www.apache.org) was configured to allow each student to use CGI programs, server-side includes, and .htaccess files to restrict access to Web pages. Perl 5.6 (www.perl.com) was installed for CGI programming, along with the Apache module for running perl scripts within the Web server (mod_perl 1.26). For server-side scripting within XHTML documents, php 4.0.6 (www.php.net) was installed and Apache was configured to recognize and process PHP scripts. The MySQL database system (mysql 3.23.41, www.mysql.com) was installed, along with the Perl and PHP modules for accessing mysql databases (perl-DBD-MySQL-1.2216 and php-mysql-4.0.6). A separate database is created for each student's use.

### Learning Outcomes

Learning outcomes are evaluated on the basis of objective products (students' final projects) and subjective evaluations (students' comments). Both support the accomplishment of the two major learning goals of the course: developing problem-solving strategies applicable to a wide range of computing tasks, and developing students' confidence in their ability to acquire and apply new skills as needed.

The final projects created for the course demonstrate that students develop technical competence in basic Web design (using XHTML and JavaScript). In many cases an ability to independently develop additional skills is also evidenced. The projects range from relatively simple Web sites and Web forms, to more complex applications that combine XHTML forms, server-side scripting, and one or more database tables to present stimuli and collect and store user input. One student's project used image maps within XHTML documents to create a prototype for a Web site demonstrating the use of a coding system for facial expressions. Several projects have used XHTML forms to collect survey data and send the data to the researcher via e-mail, and a few have added Perl CGI scripting to store data in files on the server. JavaScript has been used in several projects, including one consisting of teaching tools for statistics. Two projects have made extensive use of PHP scripting and MySQL databases, one to implement a Web-based appointment scheduling system and the other to implement an experiment on consumer decision-making. The variety of specific skills that students use to complete their projects demonstrates their ability to apply problem-solving skills to specific tasks, and their use of techniques not explicitly covered in class demonstrates students' confidence in learning and applying new skills.

Students' comments about the course have been overwhelmingly positive and are also consistent with the course objectives, with several specifically mentioning that they have applied skills from the course to their own work as researchers. One student's course project eventually led to a conference presentation (Arnott & Allbritton, 2002). Thus students' confidence in their ability to learn and apply programming skills extends beyond the context of the course.

## Advantages of Open-Source Software

The use of open-source software has contributed to the successful implementation of this course in a number of ways. On the basis of this experience, I can identify several advantages of using open-source software, and a few possible challenges.

**A model and a mechanism for collaborative knowledge construction**. The primary advantage of open-source software is that it provides a model of collaborative knowledge construction that parallels that of science (Malloy, Jensen, Regan, & Reddick, 2002). In the open-source community, knowledge (in the form of source code) is shared openly, with individual programmers building on the work of others and returning the products of their efforts to the community. A version of "peer review" also takes place, with the entire community able to offer critiques and suggest improvements. When students are introduced to open-source software, they are shown an approach to programming that is a natural extension of the problem-solving strategies and intellectual values that they learn as scientists.

The collaborative approach to problem-solving that is characteristic of the open-source software community also fits well with the learning goals of the course. Open-source software is typically developed by teams of programmers working together, and the entire community serves as a source of information and advice for solving problems through Web and discussion group postings. Students in the course learn to use such postings as a resource when working on their projects. Introducing students to the peer-group problem-solving approach of the open-source community thus provides them with a general problem-solving strategy that they can apply to many computing tasks.

**Control, freedom, and flexibility for the instructor**. Open-source software maximizes the instructor's freedom. There is no need for administrative approval for the purchase of necessary software if the university does not have an existing license. Furthermore, all open-source software can be taken for a full "test drive" to determine what best fits the instructional needs of the course.

With open-source software, it is easier to make changes as need arises. In this course, for example, I originally intended to use only Perl for server-side scripting. As I was teaching basic XHTML, however, it became evident that many of the students were likely to find it difficult to work with Perl CGI scripts. Although the Common Gateway Interface is very useful for implementing experiments on the Web (Morrow & McKee, 1998), it adds an additional layer of complexity and abstraction that is difficult for beginning programmers. To create a Web page with CGI, one must think about not only what XHTML markup would produce the desired effect, but also what Perl code would produce that markup. I decided that server-side code embedded in an XHTML page itself (such as server-side includes) would be easier for many of the students to work with. Given its ease of use and rapidly growing user base, I chose to include PHP as an alternative server-side technology along with Perl. Because I was using the open-source Apache Web server, I was able to simply download and install the required modules and in a matter of minutes reconfigure the course server to use PHP. Had I been relying on commercial software and needed to purchase a new product for such an unanticipated need, obtaining the necessary purchase orders and so forth in a timely fashion could have posed a challenge. Thus open-source software gave me the flexibility needed to adjust the tools of the course to better meet my pedagogical goals.

**Freedom and accessibility for the students**. All the software students learn to use in the course can be downloaded and installed on their own systems. There is no danger that students will learn to use a certain piece of software only to find that they have no access to it and cannot afford to purchase it themselves once the class ends. Thus, open-source software increases the likelihood that students will be able to continue to apply and develop their problem-solving skills beyond the time frame of the course.

**Increased learning opportunities for the students**. With open-source software, students have the opportunity to get a better view of what is "under the hood" and thus develop a deeper understanding of how things work. Although poring over the source code is not something students in an introductory computing course are likely to do, students in the class are introduced to the use of configuration files and log files, both of which encourage reflection on the inner workings of the software. Students can even install an open-source operating system on their own computer, and get experience administering as well as using these tools. Open-source software thus gives students additional options for continuing to develop new computing and problem-solving skills.

**Price**. My placing price last in this list of advantages is not accidental. Although the fact that open-source software can be downloaded for free is often cited as its chief selling point, the price was not a primary reason for my choosing open-source solutions for this course. From the standpoint of teaching and learning, the advantages of freedom, accessibility, flexibility, control, and especially modeling collaborative problem-solving strategies were more persuasive than cost.

## Potential Disadvantages and Challenges

**The instructor is also the system administrator**. This is often the case because (at least in my experience) most universities do not provide as much technical support for open-source software as for institutionally purchased commercial software. System administration can be time consuming, particularly when a problem arises. For example, when the course server suffered a break-in about 3 weeks before the end of the term, I had to reinstall the system, upgrade vulnerable software, restore the course and student files from backups, and formulate a new security policy to decrease the likelihood of future attacks while still allowing students sufficient access to the system. On the other hand, this experience provided a perfect opportunity to educate students about security issues in the context of a personally relevant real-world example.

**Training students to use commercial software can be more immediately applicable to their work as psychologists**, if the commercial software is more widely used than its open-source counterparts. The MySQL database system that I use in the course, for example, is not as widely used as Microsoft Access. In this case, the advantages of using MySQL has outweighed this potential disadvantage in my estimation. With MySQL, students must learn the basics of the Structured Query Language (SQL), the primary interface to almost all modern databases. Thus MySQL was more suited to accomplishing the learning goal of developing a basic understanding that could be transferred to many different database applications. Also, because MySQL is open-source, students can install it on their own computers without cost and can continue to use it in the future. Whether the advantages of an open-source solution outweigh the ubiquity of competing commercial applications as a criterion for adoption for any particular course would depend on the specific learning goals for that course.

## Conclusion

In conclusion, a course on programming and Web development designed specifically for psychology students provides students with technical and problem-solving skills that they are then able to apply to their own work and research. An open-source operating system and open-source software contribute significantly to the feasibility and success of the course, both technically in the implementation of the course and pedagogically as well.

In addition to practical and pedagogical advantages, there is also a philosophical reason for favoring open-source solutions over commercial software in a university setting. Open-source software is created collaboratively and distributed freely, with both the process and the product open to public inspection, critique, and elaboration. For institutions that see the free exchange of ideas as central to their mission, open-source software is not only a practical alternative, it is also a good fit.

## REFERENCES

Arnott, E., & Allbritton, D. W. (2002, November). *A Web-based tool for gathering ordinal rankings.* Paper presented at the the 32nd Annual Meeting of the Society for Computers in Psychology, Kansas City, MO.

Beagley, W. K. (2001). Why we need more psychology programmers/ EL Knife, a data utility for transforming spreadsheets. *Behavior Research Methods, Instruments, & Computers*, **33**, 97-101.

Birnbaum, M. H. (Ed.) (2000). *Psychological experiments on the Internet*. San Diego: Academic Press.

Birnbaum, M. H. (2001). *Introduction to behavioral research on the Internet*. Upper Saddle River, NJ: Prentice-Hall.

Dietel, H. M., Dietel, P. J., & Nieto, T. R. (2002). *Internet & World Wide Web: How to program.* Upper Saddle River, NJ: Prentice-Hall.

Dietel, H. M., Dietel, P. J., Nieto, T. R., & McPhie, D. C. (2001). *Perl: How to program.* Upper Saddle River, NJ: Prentice-Hall.

Flanagan, D. (1998). *JavaScript: The definitive guide* (3rd ed.). Sebastopol, CA: O'Reilly.

Goolkasian, P. (1997). Microcomputers in the social sciences: A new course. *Teaching of Psychology*, **24**, 204-206.

Graham, I. S. (2000). *XHTML 1.0 language and design sourcebook.* New York: Wiley.

Kučera, H., & Francis, W. N. (1967). *Computational analysis of present-day American English.* Providence, RI: Brown University Press.

Lane, D., & Williams, H. E. (2002). *Web database applications with PHP & MySQL*. Sebastopol, CA: O'Reilly.

Lerdorf, R., & Tatroe, K. (2002). *Programming PHP*. Sebastopol, CA: O'Reilly.

Malloy, T. E., Jensen, G. C., Regan, A., & Reddick, M. (2002). Open courseware and shared knowledge in higher education. *Behavior Research Methods, Instruments, & Computers*, **34**, 200-203.

Morrow, R. H., & McKee, A. J. (1998). CGI scripts: A strategy for between-subjects experimental group assignment on the World-Wide Web. *Behavior Research Methods, Instruments, & Computers*, **30**, 306-308.

Reese, G., Yarger, R. J., & King, T. (2002). *Managing & using MySQL: Open source SQL databases for managing information & Web sites* (2nd ed.). Sebastopol, CA: O'Reilly.

Sargent, D. M. (1996). On-line computers in psychology: A laboratory course for advanced psychology majors. *Behavior Research Methods, Instruments, & Computers*, **28**, 354-355.

Schwartz, R. L., & Phoenix, T. (2001). *Learning Perl* (3rd ed.). Sebastopol, CA: O'Reilly.

Siever, E., Spainhour, S., Hekman, J. P., & Figgins, S. (2000). *Linux in a nutshell* (3rd ed.). Sebastopol, CA: O'Reilly.

Wall, L., Christiansen, T., & Orwant, J. (2000). *Programming Perl* (3rd ed.). Sebastopol, CA: O'Reilly.

Welsh, M., Dalheimer, M. K., & Kaufman, L. (1999). *Running Linux* (3rd ed.). Sebastopol, CA: O'Reilly.

(Manuscript received November 19, 2002;
revision accepted for publication January 15, 2003.)