Behavior Research Methods
2007, 39 (3), 471-481

Web-MCQ: A set of methods and freely available open source code for administering online multiple choice question assessments

Claire Hewson
University of Bolton, Bolton, Lancashire, England

E-learning approaches have received increasing attention in recent years. Accordingly, a number of tools have become available to assist the nonexpert computer user in constructing and managing virtual learning environments, and implementing computer-based and/or online procedures to support pedagogy. Both commercial and free packages are now available, with new developments emerging periodically. Commercial products have the advantage of being comprehensive and reliable, but tend to require substantial financial investment and are not always transparent to use. They may also restrict pedagogical choices due to their predetermined ranges of functionality. With these issues in mind, several authors have argued for the pedagogical benefits of developing freely available, open source e-learning resources, which can be shared and further developed within a community of educational practitioners. The present paper supports this objective by presenting a set of methods, along with supporting freely available, downloadable, open source programming code, to allow administration of online multiple choice question assessments to students.

With the rapidly increasing interest in e-learning strategies and techniques in higher education, a number of tools are becoming available to aid the design and implementation of Web-based teaching, learning and assessment procedures. The present article describes a set of methods and supporting programming code, developed at the University of Bolton, to support administration of online MCQ assessments to psychology undergraduates. The code is freely available online (www.clairehewson.co.uk/LTSNproject/MCQtool) as a set of downloadable, customizable, open source files. Although a range of commercial e-learning software packages are now available, with fairly sophisticated ranges of functionality (WebCT and Blackboard[1] being probably the most comprehensive "virtual learning environment" packages currently available), several authors have advocated encouraging the development of freely available, open source, software (e.g., Gordon & Malloy, 2002; Malloy, Jensen, Regan, & Reddick, 2002). Aside from the fact that commercial packages are typically expensive (Dempster, 1998; Henly, 2003), and not always easy to learn to use (Henly, 2003), it has been argued that the constraints imposed by reliance on packages developed by a relatively small number of experts, with commercial rather than pedagogical interests, can be detrimental to the teaching and learning process (Gordon & Malloy, 2002; Malloy et al., 2002). As Gordon and Malloy (2002) comment, "it is better for software to meet the pedagogical needs of the instructor than it is for the instructor to have to shape his or her pedagogy to fit the limitations of the tool" (Gordon & Malloy, 2002, p. 241). Malloy et al. refer to the "open knowledge initiative" (website: Web.mit.edu/oki/) which aims to provide a "collaborative community of academics who are developing on-line teaching software" (p. 202). Thus, while commercial packages such as those mentioned above are created with the intention of making e-learning practices readily available to the nonexpert computer user, and may certainly be useful in this respect, they may also incur disadvantages and impose restrictions.

The set of methods and associated code described in this paper are offered as a free resource which will be of use to those wishing to implement either formative or summative MCQ assessments online. Advantages of online assessment include automated scoring and storing of students' responses, which can lead to enhanced efficiency, quicker turnaround time, and reduced potential for human error; pedagogical benefits of providing immediate automated feedback; convenient access to and submission of assignments; support for distance learning and widening participation initiatives. While educators have been fairly receptive to the use of online *formative* assessment procedures (e.g., Buchanan, 2000; Henly, 2003), and have found these to be beneficial (e.g., Henly, 2003; Van Hoof & Porteous, 2004), adoption of online *summative* assessment practices has raised a greater degree of concern due to reliability and validity issues (e.g., Roy & Armarego, 2003). Reliability may be threatened due to technical problems which can lead to system per-

C. Hewson, ch5@bolton.ac.uk

formance failures; for example, a server error may cause an assessment to go offline, or answers not to be submitted properly. Validity concerns arise because other factors, besides actual levels of competency in what is being tested, may impinge upon performance on online assessments. Such factors could include things like levels of computing experience, computer attitudes (e.g., anxiety, engagement, etc.) and potential access to correct answers by respondents due to security weaknesses. Difficulties in verifying a respondent's identity may also threaten the validity of online assessments (though this can also be a problem for some offline assessment methods, of course).

Such concerns are nontrivial, and further research is needed to verify the extent to which online summative assessment procedures can provide valid, reliable measures of levels of competency in the relevant subject area, compared with their offline equivalents. Preliminary support for the validity of the procedures described here has been provided by Hewson, Charlton, and Brosnan (2007). These authors compared online and offline administrations of the same (summative) MCQ assessment (using the third implementation described in the present paper), and examined whether performance in either context was related to levels of computer anxiety and/or engagement. Overall, they report that students taking the assessment online did not perform any worse than those taking the same assessment offline (in pen and paper format), and that performance on either the online or offline assessment was not affected by levels of computer anxiety or engagement. These results are promising, though further research is needed to clarify the generalizability of these findings across different assessment contexts (e.g., course content), settings (e.g., in-class as opposed to "truly" online), and student cohorts (e.g., those with limited levels of computer-related experience, or high levels of computer anxiety). The ready availability of the programming code to support the procedures described here, and evaluated in the aforementioned study, will help facilitate such follow-up studies.

It is beyond the scope of the current paper to review in any depth the online assessment tools already available, or to compare the functionalities of these and the present Web-MCQ resource. As well as the commercially available Virtual Learning Environment (VLE) packages mentioned above, other more dedicated commercial packages for developing online assessments are on offer (e.g., Question Mark's Perception, see www.qmark.com/perception). Several free software packages capable of supporting online assessments have also been developed, including CASTLE toolkit (www.le.ac.uk/castle/index.html); CALnet (www.webecon.bris.ac.uk/calnet/); Hot Potatoes (hotpot.uvic.ca/); TOIA (www.toia.ac.uk/); Gordon and Malloy's (2002) On-line Homework/Quiz/Exam applet (www.psych.utah.edu/learn/homework); White and Hammer's (2000) Quiz-o-Matic (www.depaul.edu/~rwhite1/quiz). At the time of writing, however, both of the latter two packages were inaccessible.[2] Indeed, it is not so improbable for both commercial and free packages to become unavailable as new developments emerge, or as supporting funding expires, which is another reason for encouraging the development of a community of practitioners who are committed to sharing and developing free, open source, teaching and learning software resources.

WEB-MCQ: DESCRIPTION, FUNCTIONALITY AND REQUIREMENTS

Web-MCQ consists of a number of customizable files provided for download at www.clairehewson.co.uk/LTSNproject/MCQtool. Since it was initially developed, the resource has evolved in a number of ways to meet a range of needs. Here three different implementations are described, each with slightly different functions, ranging from a procedure which can be implemented with minimal levels of technical equipment and expertise (useful for administering formative assessments which provide immediate feedback), to a more sophisticated implementation which requires access to a server on which CGI scripts can be run (which allows for enhanced reliability and security, and enables data storage, thus creating scope for administering online summative assessments). The following sections describe each implementation in more detail and explain how to download, edit and install the relevant files from the Web-MCQ website. A certain basic level of knowledge of how to open, edit and save (HTML) files, and upload these to a Web server, is assumed here. The User Manual provided on the website provides a more detailed step-by-step guide. The code provided may be freely used and customized to suit individual needs.

1. Formative Assessment with Automatic Feedback

Implementation. This implementation presents the respondent with a set of twenty MCQs and provides automatic feedback in terms of an overall score. The respondent's answers and total score are not stored in this version. Implementation simply involves saving the file 'MCQquiz.txt' from the Web-MCQ website and customizing it as described below. The quiz can be run on a local computer, or can be placed on a Web server for access via the world wide Web (WWW).

Once saved, the file MCQquiz.txt should be opened in a suitable text editor so that the HTML code (a shortened version of which, with only one MCQ, is shown in Table 1) can be customized.

If the code displays without line breaks (which may happen if opened using certain text editors, e.g., *Notepad*), it should be opened using a different text editor. Microsoft's *Wordpad* should display the code correctly. Figure 1 shows how the code in Appendix A displays in a Web browser (achieved by renaming the text file with an .html extension, i.e., MCQquiz.html, and opening in any browser).

Customizing the question and answer content is simply a matter of replacing the relevant text, i.e., "Insert question 1 here," and so on, with the desired content. Scoring is carried out by a javascript routine in the HTML form: **function score()**. This works as follows. The correct answers are coded in lines like this one: **if(document.myform[1].checked = = true){x = x + 1}**. These lines specify the correct answers as elements in the "form elements array,"

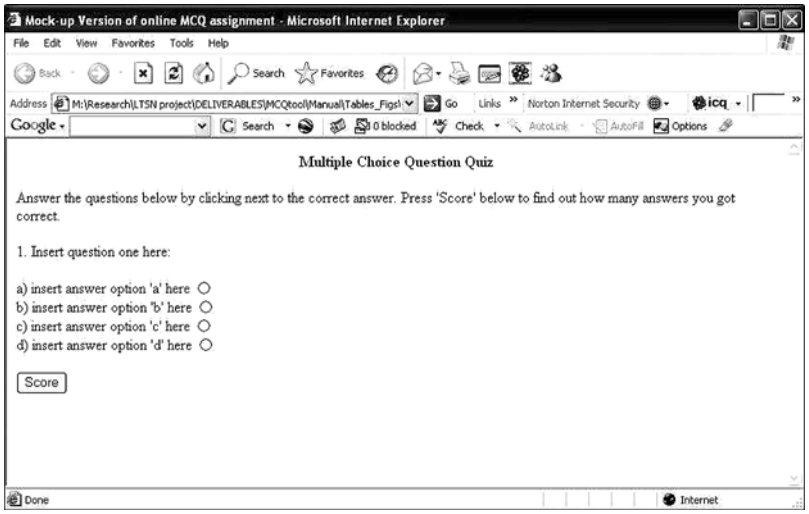

Figure 1. MCQquiz.html (shortened code) as displayed in a Web browser.

each form element (e.g., radio button, text input box, check box) being identified by a number indicating it's position within this array. The number specifying the form element is that which appears in square brackets (above). In this case form element 1 has been specified, which here corresponds to the first radio button input in the form, i.e., question 1, answer a. The second form element, element 2, would refer to question 1, answer b. The third to question 1, answer c, and so on. MCQquiz.txt specifies answer a as the correct answer to all questions (i.e., elements 1, 5, 9, 13, etc.). Once all the correct answers have been specified as form elements, in lines of code as shown above, the javascript routine works by checking whether each of these elements has been selected, and for each that has adds 1 to the value of x, where x is a variable starting with a value of zero. It should be apparent now how x will end up being assigned a value equal to the number of correct answers selected by a respondent, that is, their total score. Thus, specifying the correct answers in MCQquiz.txt is simply a matter of editing the numbers in square brackets. Note that in order for the scoring procedure (or indeed any javascript routine) to work the respondent must have javascript enabled in their browser.[3] Of course, to make the MCQ quiz available on the WWW it must be uploaded to a Web server.

Evaluation. This formative assessment implementation with automatic feedback may be useful for providing online practice quizzes, the use of which has been demonstrated to have pedagogical benefits. Also, due to its lack of technological sophistication, it can be easily implemented by users with modest levels of technical expertise. For more experienced users, the code may be customized to include additional features, such as providing the correct answers during feedback (though, arguably, allowing repeated attempts without correct answers being revealed could prove more pedagogically advantageous). This implementation is limited, however, in terms of not enabling the storing of responses, and in coding the correct answers to questions within the HTML form itself (thus providing scope for a computer-literate respondent to detect and decode these answers). The next implementation involves a simple adaptation of MCQquiz.txt, which allows responses to be stored, thus making it potentially useful for administering online summative assessments.

2. Summative Assessment Using E-mail to Store Responses

Implementation. This implementation functions as above, except that instead of providing automatic feedback to respondents, the answers to individual questions, and the total score, are sent to a specified e-mail address. An ID-checking function is also incorporated. This implementation involves accessing and customizing the file MCQtest-email.txt, available from the Web-MCQ website. The (shortened) HTML code is displayed in Appendix B.

Question content and correct answer codings should be customized as described in the previous implementation. In coding the correct answers, however, note that because two additional respondent inputs have been added at the start of the form, asking for the respondent's user ID and name, these are assigned the form element values "1" and "2," respectively, in the form element array. Question 1, answer a, thus becomes form element "3" in this HTML code. A javascript routine is used to manage the ID-checking procedure. This works by listing all the valid IDs which may be accepted, and alerting the user with an error message ("You have not entered a valid ID. Please try again and if the problem persists contact the module tutor") if an ID not in this list is entered into the ID form element. Of course, the invalid ID alert message can be edited as desired. The range of IDs which will be accepted can be customized by editing as appropriate the current values "userID1," "userID2," and "userID3" in the HTML form. The text within the quotes specifies exactly what the user must enter, and is case sensitive. If a correct ID is entered then "ID verified" will be returned upon clicking the button as instructed. As many IDs as required can be included by adding more identical lines of code as

appropriate, e.g., **&& (document.myform.ID.value ! = "userID4").**

When the respondent clicks the "submit assignment" button the ID-checking routine is invoked; if a valid ID is not detected then the form will not be submitted (and the error alert will be presented).

The final edit needed involves specifying an e-mail address to which the form data should be sent. This is done by replacing the fictitious address (myemail@myserver .ac.uk) in the **action = "mailto:** command with a valid e-mail address. The e-mail message sent when the respondent submits the form will contain the data in the following format: ID = userID1&name = &Q1 = a&totalscore = 1. In this case the respondent entered "userID1," did not enter a name, answered "a" to question 1 (but did not answer any other questions), and obtained a total score of 1. Obviously this data will need some tidying up before entering into a spreadsheet or data analysis package. In order for the "mailto:" command to function correctly, the respondent will need an e-mail client (e.g., Outlook, Pegasus, etc.) correctly installed and configured on their computer, and an active Internet connection.

Evaluation. The above implementation provides a relatively straightforward and effective way of administering MCQ assessments which includes ID-checking, automated scoring, and storing of respondents' answers within the body of an e-mail message sent to an appropriate e-mail account. However, there are some drawbacks to this approach. First, the data is not stored in an easily readable format, and would need to be tidied up somewhat before being imported into a data processing package. Second, and more crucially, some mailers will open up a window displaying the contents of the e-mail message to be sent—in this case, the respondent's answers and total score—thus allowing the user to view the contents of the e-mail and consider whether they really want to send it. Obviously, this is problematic in this context, and may lead to attempts to either edit the score value directly (which of course can be detected if the tutor compares the total score against responses to individual questions), or to make multiple attempts, perhaps by trial and error, until a satisfactory score is achieved and sent. This makes the approach unworkable for summative assessments unless the tutor takes measures to maintain control over the range of browsers/e-mail clients[4] which students may use (e.g., by requiring the assessment to be taken in-class), or removes the automatic scoring function (which can be easily done).[5] Having the scoring key within the HTML form itself is less than ideal anyway, whether or not the mailer displays answers before sending, as respondents with programming experience may be able to detect the correct answers by viewing the source code (done by clicking Page Source under the browser's View menu). A final issue concerns the need for the respondent to have an e-mail client correctly installed and configured on their computer in order for the answers to be sent by e-mail; while most users do have this, some (e.g., those relying on "webmail") may not.

Given the above reliability and validity issues, an alternative implementation is now described which makes use of a *CGI script* (a program residing on a Web server that executes and processes the data submitted by a user through an HTML form) to process and save incoming data from respondents. Since this implementation moves the scoring process from the HTML form to the CGI script, it increases the level of security over the correct answers, while maintaining the convenience of using automated scoring. It also increases reliability by not relying on the respondent having an e-mail client appropriately configured on their machine, and stores information about the respondent's browser type, IP address, and date and time of submission. It does, however, require access to a server space which allows execution of CGI scripts. The script provided here is written in Perl; most modern servers can run Perl CGI scripts, including the widely used open-source UNIX-based Apache, and many Windows-based servers. The current script saves the respondent's data to a specified file, in a format (CSV, or "comma separated values") ready for import into SPSS or a similar data analysis package, and also sends the data in an easily readable format in the body of an e-mail message to a specified e-mail address.

3. Summative Assessment Using a CGI Script to Process Submitted Data

Implementation. This implementation makes use of the files MCQtest-cgi.txt and MCQscript.pl, which can be saved from the Web-MCQ website. MCQtest-cgi.txt should first be edited to customize the question content and ID codes (as described above). Note that the scoring function has been omitted from the HTML form in this implementation, so there is no need to specify any correct answer codings here. Other than this, the HTML code is very similar to that of MCQtest-email.txt (Appendix B). The HTML code then needs to be edited to specify the location and name of the CGI script which will process the data when the respondent submits their answers. This is done in the **action = "** command. Note that in the previous implementation this command specified an e-mail address to which data was to be sent. The code in MCQtest-cgi.txt currently specifies the location of the CGI script as a fictitious Web address: http://www.servername .com/cgi-bin/MCQscript.pl. That is, the address of the server (http://www.servername.com), and the directory on that server (cgi-bin) in which the script (MCQscript.pl) resides. This needs to be replaced with the location of the CGI script on an appropriate actual server. Typically, a CGI script will need to be placed in a special directory on the server called "bin" or "cgi-bin," in which CGI scripts are allowed to execute. If the HTML form is also placed in this same directory then all that needs to go in the **action = "** command is the name of the script itself (e.g., MCQscript.pl) and the HTML form will find it. Otherwise the appropriate full path to the script will need to be specified. The Web server administrator should be able to advise on the appropriate directory in which to place CGI scripts, and the full path name to the script.

The CGI script MCQscript.pl then needs to be customized (by opening and editing within a text editor). The full code for MCQtest.pl is shown in Appendix C. First, an appropriate mail server address needs to be specified (currently mail.bolton.ac.uk). In the line below, the e-mail

address to which the data should be sent is specified (currently myemail@myserver.ac.uk). Below this, the name of the file in which data will be stored is given (here MCQtest.dat). If the **$mail_recipient** field is left blank then no e-mail will be sent, but the script should still execute and save data to the specified file. However, if the mail server and data file fields are left blank an error will be returned and the script will fail to execute correctly. The data file itself (simply created as an empty text file) will need to be stored somewhere on the Web server, and the appropriate full path name specified (if it does not reside in the same directory as the CGI script, as assumed in the current example). The **$mail_sender** string, here MCQ assignment, can be changed as desired. The **$debrief_url** specifies a page to which the respondent's browser is automatically directed after answers have been submitted. Finally, the correct answers for the questions are specified by editing the values in the correct answers hash (i.e., **"Q1" = > "d,"** etc.), as appropriate; also the line of code at the top of page 10 (Appendix C) (**$smtp->mail('Webform@mail.bolton.ac.uk');**), needs to be changed so that the mail server specified (here mail.bolton.ac.uk) is the same as that which was specified earlier. The rest of the script does not need to be modified (though it can, by anyone with some knowledge of Perl).

To make the MCQ assessment available online, the HTML form, CGI script and debrief page need to be placed on a Web server, and their permissions set appropriately. The HTML form (MCQtest-cgi.html) does not require any special permissions. The CGI script (MCQtest.pl) should be placed in a CGI-enabled directory (checking that this location has been correctly specified in the HTML form), and given execute permission for all users. The data file needs to be given write access for all users, in order for data to be saved. With all the above in place, the HTML form should submit the data to the CGI script, which will process it and save it to the specified data file, as well as send it in the body of an e-mail message to the address specified. After a respondent has submitted data, the data file will contain a line which looks something like this:

```
Mon Mar 28 18:52:24
2005,193.63.49.30,userID1,Hammy,a,b,a,a,d,b,a,,,,,,,0
```

This data is in CSV format, which can be imported directly into a data analysis package such as SPSS. The e-mail message will display the above data as shown in Appendix D. This respondent (Hammy) answered questions one to seven and left the remainder blank, obtaining a total score of zero.

Evaluation. The above approach has a number of advantages, as have been outlined already. It does, however, require greater levels of technological expertise and resources, compared with both of the previous approaches described in this paper.

4. Further Refinements

Providing Immediate Feedback and Storing Responses. It may, of course, be useful to provide immediate feedback to students *and* store their responses. This can be easily achieved by adapting the implementation just described; the CGI script can be modified so as to post back a response when data is submitted which announces the total score obtained. This is achieved by editing the final lines of code in the CGI script which print information back to the respondent and redirect their browser to another Web page. For example, editing the lines which include the text stating "Your answers are being sent" as shown below will provide a receipt for the respondent which states their name, ID, date and time of submission, and total score:

```
print " <font face = \"Verdana, Arial, Helvetica,
sans-serif\" color = \"#000000\" > <b> Your
MCQ assignment has been received. You can
print out this receipt for your records. <P>
Name: $FORM{name} <BR> Student number:
$FORM{ID} <BR> Date received: $date_
time. <P> <BR> You have scored $score correct
out of 20. <p> \n";
```

Of course, the total score may or may not be included here, as appropriate (to omit, just delete the final sentence "You have scored $score correct out of 20." in the above code). If using the code suggested above, the line: **print "<meta http-equiv = Refresh\ content = \"3;url = $debrief_url\" >\n";** should be deleted (or commented out by placing a # in front of it) so that the respondent's browser is not automatically redirected to another webpage, else they may miss digesting the information provided and most certainly will not have time to print out their receipt.

Conclusion, Usage, Limitations

The set of methods and supporting implementation code described here provide scope for administering online MCQ assessments to students, and scoring and/or storing responses automatically. This may have the benefits of time and cost savings, as well as convenience for students, who can access and submit the assessment from a range of locations (bearing in mind that while most students these days have ready access to a computer, some may not). In it's simplest form, the formative version can be used as an online "quiz" which provides automatic feedback and is easy to implement. If storing of responses is required, this can be done without any need for CGI scripting, using the "mailto:" command, though there are various drawbacks to this approach, as discussed above. Given the extent of these problems, in the current context, using this approach to implement "truly" online assessments (i.e., those in which the respondent may access and submit the assessment from any WWW-enabled computer) becomes untenable, unless the automatic scoring feature is disabled, which then removes one of the key benefits of the approach. Even then, the scope for knowledgeable respondents to detect the correct answers poses a serious threat to data validity, which should discourage such summative use of this implementation. Usage in a controlled, in-class, exam setting is feasible, however.

A more effective solution for administering online summative assessments involves using the third imple-

mentation described above. This resolves the key issues identified in using implementation 2. However, this does require access to a server which allows CGI scripts to run, which may not always be practicable, or readily available to all users.

There is scope for developing the current tool in various ways. For example, one possible issue is the way IDs are stored within the HTML form itself, which leaves open the possibility of respondents gaining access to this information and selecting and using any of these codes to be able to submit the assignment (though it is difficult to think of a reason why a student might want to adopt this strategy). To address this, the ID-checking function could be moved into the CGI-script with a little extra programming work; ID codes could be stored within the script itself or in a database which the script is able to read. Using an approach in which IDs are loaded from a database may also help save time, as it could prove quite cumbersome (especially where the pool of users is large) to have to list manually all user IDs in the HTML form, as suggested in the current implementations. One may also want to include further text input boxes within the HTML form, e.g., asking for an e-mail address to be provided, or change the layout, font size, number of questions, response format (radio buttons, check boxes, text input boxes), and so on. Users already familiar with HTML will be able to customize the code to their liking; those new to HTML may find one of the many introductory guides available useful (e.g., Castro, 2002; www.cs.cf.ac.uk/Dave/PERL/node249.html). The scope for implementing such modifications is a key advantage of free, open source software, compared with the more restricted commercially available packages.

A more ambitious development of the current resource would be to implement a graphical user interface (GUI), which would make it more readily accessible to users with lower levels of computer expertise. This approach would not preclude making the source code available for modification by more experienced users, who may desire enhanced levels of customizability. However, this development would imply increased complexity of the tool, requiring the use of an advanced programming language such as Java, and this would have implications for the ease with which more novice users could learn to customize features of the tool to suit their needs. Also, unless use of the tool was hosted on a third-party Web server, knowledge of how to install the relevant files on a local Web server would still be required.

To conclude, the aim here has been to provide a simple yet effective resource which allows users with modest levels of computer expertise to implement MCQ assessments online. A further aim has been to make available fully customizable, open source code, so that individual users with moderate levels of computer programming experience may adapt this resource to suit their own needs.

AUTHOR NOTE

The Higher Education Academy (formerly LTSN) provided funding to support the development and evaluation of the current Web-MCQ resource. The author wishes to acknowledge the valuable assistance and expertise of Peter Yule, who worked on this project to help in developing the implementations described here. Thanks also goes to an anonymous reviewer who provided helpful comments on an earlier version of this manuscript. Correspondence concerning this article should be addressed to Claire Hewson, Department of Psychology and Life Sciences, University of Bolton, Deane Rd., Bolton, Lancashire, BL3 5AB, England (e-mail: ch5@bolton.ac.uk).

REFERENCES

BUCHANAN, T. (2000). The efficacy of WWW-mediated formative assessment. *Journal of Computer Assisted Learning*, **16**, 193-200.

CASTRO, E. (2002). *HTML 4 the world wide Web with XHTML and CSS: Visual quickstart guide* (5th ed.). Peachpit Press.

DEMPSTER, J. (1998). Web-based assessment software: Fit for purpose or squeeze to fit? *Interactions*, **2**. Online Journal, University of Warwick. Retrieved, June 13, 2006 from www.warwick.ac.uk/ETS/interactions/vol2no3/dempster.htm.

GORDON, O. E., & MALLOY, T. E. (2002). On-line Home homework/quiz/exam applet: Freely available Java software for evaluating performance on line. *Behavior Research Methods, Instruments, & Computers*, **34**, 241-244.

HENLY, D. C. (2003). Use of Web-based formative assessment to support student learning in a metabolism/nutrition unit. *European Journal of Dental Education*, **7**, 116-122.

HEWSON, C., CHARLTON, J., & BROSNAN, M. (2007). Comparing online and offline administrations of multiple choice question assessments to psychology undergraduates: Does assessment modality or computer attitudes influence performance? *Psychology Learning and Teaching*, **6**, 37-46.

KRAAN, W. (2005). Blackboard to acquire WebCT. CETIS. Retrieved, June 13, 2006 from www.cetis.ac.uk/content2/20051014001848.

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.

ROY, G. G., & ARMAREGO, J. (2003). The development of on-line tests based on multiple choice questions. In D. Taniar and W. Rahayu (Eds.), *Web-Powered Databases* (Hershey [Penn]: Idea Group, pp. 121-143). Retrieved, June 13, 2006 from http://eng.murdoch.edu.au/~jocelyn/papers/WPDBv2.pdf.

VAN HOOFF, H., & PORTEUOUS, L. (2004, November). The efficacy of on-line formative assessment exercises (short report). *LTSN Newsletter*.

WHITE, R. J., & HAMMER, C. A. (2000). Quiz-o-Matic: A free Web-based tool for construction of self-scoring on-line quizzes. *Behavior Research Methods, Instruments, & Computers*, **32**, 250-253.

NOTES

1. At the time of writing, Blackboard is in the process of taking over WebCT (Kraan, 2005).

2. The former due to the browser repeatedly freezing upon trying to load the java applet provided on the webpage, the latter being no longer available at the location cited (the current author also had no luck locating the Quiz-o-Matic tool described in White & Hammer (2000) by e-mailing the contact address provided, e-mailing the support team at that address, or conducting a Web search. The Web search did, however, generate what appeared to be a previous more basic "Quiz-o-Matic '76" version of this tool).

3. It is also possible that certain security settings may cause the scoring routine to fail, in which case these settings will need to be adjusted as appropriate.

4. On testing, Mozilla's (version 1.6) e-mail client did open up an e-mail window displaying the answers and total score, when configured with a POP3 mail server account. When configured for an IMAP mail server account, this same mailer failed to include any data in the body of the e-mail message, with only an empty e-mail message being sent. Outlook Express's e-mail client included the answers and total score as a file attachment to an e-mail message, which could be opened and viewed from within the user's outbox prior to sending (thus generating the same problem as described previously of the user being able to view their total score prior to sending).

5. This can be done by removing all the code from **function score()** down to and including the **}** just before **/script**, and then editing the line below **/script**, i.e., **<form method = post name = myform action =**

"mailto:myemail@myserver.ac.uk" onsubmit = "return score()"> so that **onsubmit = "return score()">** becomes *onsubmit = "return checkform*()">. Finally, the third line from the bottom of the code, i.e., **<input type = submit value = "submit assignment"> <input type = hidden name = totalscore value = "">**, should be edited to read: <*input type = submit value = "submit assignment*"> (i.e., remove the latter part within < >). This done, the "totalscore = " line in the e-mail message will no longer appear.

APPENDIX A
MCQquiz.txt (Shortened HTML Code)

```
<html> <head> <basefont size = 3>
<title> Mock-up Version of online MCQ assignment </title>
<script language = javascript>
  function score()
{
  var x = 0
  var y = 20
  if(document.myform[1].checked = = true){x = x + 1}
{ alert("Your score is " + x + " out of " + y)}
document.myform.totalscore.value = x
  return true;
}
</script>
<form method = post name = myform onsubmit = "score()">
<input type = "hidden">
<P> <center> <B> Multiple Choice Question Quiz <BR>
<BR> <P>
</B> </center>
Answer the questions below by clicking next to the correct answer. Press 'Score' below to find out
how many answers you got correct.
<BR> <P>
1. Insert question one here: <P>
a) insert answer option 'a' here <input type = radio name = "Q1" value = "a"> <br>
b) insert answer option 'b' here <input type = radio name = "Q1" value = "b"> <br>
c) insert answer option 'c' here <input type = radio name = "Q1" value = "c"> <br>
d) insert answer option 'd' here <input type = radio name = "Q1" value = "d"> <br>
<BR> <P>
<P>
<input type = button value = "Score" OnClick = "score()">
</form>
</html>
```

APPENDIX B
MCQtest-email.txt (Shortened HTML Code)

```
<html> <head> <basefont size = 3>
<title > MCQ Assignment </title>
<script language = javascript>
function checkform()
{
  if ((document.myform.ID.value ! = "userID1")
&& (document.myform.ID.value ! = "userID2")
&& (document.myform.ID.value ! = "userID3")
    )
  {
   // something is wrong
   alert('You have not entered a valid ID. Please try again and if the problem persists contact the
module tutor');
    return false;
  }
else
{alert('ID verified')
}
  return true;
}
function score() {
  var x = 0
  var y = 3
  if(document.myform[3].checked = = true){x = x + 1}
document.myform.totalscore.value = x
  return checkform()
}
</script>
<form method = post name = myform action = "mailto:myemail@myserver.ac.uk" onsubmit =
"return score()">
<P> <center> <B> MCQ Assignment <BR>
Date due: 24th March, 2005 <P>
</B> </center>
<P> Please type in your student ID number and click on 'Verify' <P> <input type = "text" name =
"ID">
<input type = "button" value = "Verify" onclick = "return checkform()">
<P> <BR>
Name: <input type = "text" name = "name">
<BR> <BR> <P>
Please answer the questions below by clicking next to the correct answer. Answer all questions. <B>
Do not press the 'Enter' (Return) key on your keyboard while still completing this assignment as this
will cause your answers to be automatically sent before completion. </B>
<P> When you are sure you have completed all the questions please press the <B> 'send assign-
ment' </B> button to send your completed answers. <BR> <BR> <P>
1. Insert question one here: <P>
a) insert answer option 'a' here       <input type = radio name = "Q1" value = "a"> <br>
b) insert answer option 'b'  here <input type = radio name = "Q1" value = "b"> <br>
c) insert answer option 'c' here       <input type = radio name = "Q1" value = "c"> <br>
d) insert answer option 'd' here       <input type = radio name = "Q1" value = "d"> <br>
<BR> <P>
THAT IS THE END OF THE ASSIGNMENT. <P>
<P>
<input type = submit value = "submit assignment"> <input type = hidden name = totalscore
value = """>
</form>
</html>
```

APPENDIX C
The CGI Script MCQtest.pl

```
#!/usr/bin/perl
#use CGI::Carp qw(fatalsToBrowser);
use Net::SMTP;
#use warnings;
# define parameters for your application
# Mail params
$mail_server = 'mail.bolton.ac.uk';
$mail_recipient = 'my.email@myserver.ac.uk';
$data_file = 'MCQtest.dat';
$mail_sender = 'MCQ_assignment';
$debrief_url = 'http://www.servername.ac.uk/username/MCQdebrief.html';
# create array of question names, to specify the output formatting order
@itemnames = (
  "Q1," "Q2," "Q3," "Q4,"
  "Q5," "Q6," "Q7," "Q8,"
  "Q9," "Q10," "Q11," "Q12,"
  "Q13," "Q14," "Q15," "Q16,"
  "Q17," "Q18," "Q19," "Q20"
  );
# create hash of correct answers for your questions, for scoring
%itemanswers = (
    "Q1" = > "d," "Q2" = > "c," "Q3" = > "d," "Q4" = > "b,"
    "Q5" = > "c," "Q6" = > "d," "Q7" = > "d," "Q8" = > "d,"
    "Q9" = > "c," "Q10" = > "a," "Q11" = > "d," "Q12" = > "c,"
    "Q13" = > "a," "Q14" = > "a," "Q15" = > "d," "Q16" = > "c,"
    "Q17" = > "b," "Q18" = > "c," "Q19" = > "d," "Q20" = > "a"
    );
# this might work to avoid defining @itemnames above, but it reorders keys
#@itemnames = keys %itemanswers;
# recover time and hostname
$date_time = localtime;
$remote_address = $ENV{'REMOTE_HOST'};
if (!$remote_address) { $remote_address = $ENV{'REMOTE_ADDR'}; }
$forwarded_for =
$ENV{'HTTP_X_FORWARDED_FOR'};
if ($forwarded_for) { $remote_address = $forwarded_for; }
# read the input data (method = POST)
read(STDIN, $formdat,
$ENV{'CONTENT_LENGTH'});
# split the input data into individual form items
@namevals = split(/&/,$formdat);
# create a hash containing the form data
foreach $pair (@namevals) {
  # separate into name and value
  ($name,$value) = split(/ = /,$pair);
  # convert + signs to spaces
  $value = ~ tr/ + / /;
  # convert hex pairs (%HH) to ASCII
  $value = ~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C," hex($1))/eg;
  # store vals in a hash called %FORM
  $FORM{$name} = $value;
}
# now, prepare the outputs (file and email)
open(SURVEY_FILE," + > > $data_file") or die "$^E";
print SURVEY_FILE "$date_time,$remote_address,";
# email code
$smtp = Net::SMTP-> new($mail_server) || die ^E;
```

APPENDIX C (Continued)

```
$smtp-> mail('web_form@mail.bolton.ac.uk');
$smtp-> to($mail_recipient);
$smtp-> data();
$smtp-> datasend("To: $mail_recipient\n");
$smtp-> datasend("From: $mail_sender\n");
$smtp-> datasend("\n");
$smtp-> datasend("Date\: $date_time \n");
$smtp-> datasend("Remote Address\: $remote_address \n");
# print out the ID and name values to file...
print SURVEY_FILE
"$FORM{ID},$FORM{name},";
# ...and to email
$smtp-> datasend("ID\: $FORM{ID} \n");
$smtp-> datasend("Name\: $FORM{name} \n");
# initialize the score
$score = 0;
# write one value (possibly empty) to file for each question item
# even if it doesn't appear in the form submission
# also, do scoring during the same pass
foreach $thename (@itemnames) {
  # score the answer (undef values don't get counted)
  # add 1 point for each correct answer, no penalty for incorrect
  if($itemanswers{$thename} eq
$FORM{$thename}) {
  $score + + ;
  }
  # print comma-delimited value (possibly empty) to file...
  print SURVEY_FILE "$FORM{$thename},";
  # ...and name: value pair to email
  $smtp-> datasend("$thename\:
$FORM{$thename} \n");
}
# after the raw data, print out the score too
print SURVEY_FILE "$score\n";
$smtp-> datasend("Score\: $score \n");
# data processing is over, so close outputs
close (SURVEY_FILE);
$smtp-> dataend();
$smtp-> quit;
# finally, print message page back to user
print "Content-type: text/html\n\n";
print "<html> \n";
print "<head> \n";
print "<meta http-equiv = Refresh\ content =
\"3;url = $debrief_url\">\n";
print "<title> User Survey </title> \n";
print "</head> ";
print "<body bgcolor = \"white\">";
print "<center> \n";
print "<br> <br> <br> \n";
print "<font face = \"Verdana, Arial, Helvetica, sans-serif\" color = \"#000000\"> <b> Your an-
swers are being sent <P> <BR> Please wait... <br> <p> \n";
print " </center> ";
print " </body> \n";
print " </html> \n";
```

APPENDIX D
Contents of the E-mail Message Sent by the CGI Script (MCQtest.pl)

Date: Mon Mar 28 18:52:24 2005
Remote Address: 193.63.49.30
ID: userID1
Name: Hammy
Q1: a
Q2: b
Q3: a
Q4: a
Q5: d
Q6: b
Q7: a
Q8:
Q9:
Q10:
Q11:
Q12:
Q13:
Q14:
Q15:
Q16:
Q17:
Q18:
Q19:
Q20:
Score: 0

(Manuscript received January 7, 2006;
revision accepted for publication May 8, 2006.)