CHAPTER V

THE PROGRAMMING OF RANDOMIZED PRACTICE PROBLEMS

The past decade has clearly demonstrated that classroom educators are quite resourceful in utilizing the Internet to improve the learning environment (Lever-Duffy & McDonald, 2008). From their own professional enhancement of the searching and sharing of lesson plans, to extending the classroom beyond traditional barriers by posting resources for students and parents on the teacher’s personal website, the Internet is a cornucopian tool and the use of online assessment tools only further expands that bounty. However, the types of online assessment tools vary considerably in programming, delivery, functionality, and cost. On the lower end of this continuum, educators can utilize a single, static webpage that guides a student through various questions. The student works their way through the content at their own pace, either keeping their thoughts and ideas in working memory or writing their answers down on scrap paper, finally self-checking their work with a solution key located at the bottom of the page. At the upper end, a course/content management system can provide a more robust structure for an entire school district. Each educator can develop and share content, including a bank of questions that can be fed into individual assessments. This option utilizes a server to both serve and record data – creating a wide range of options in capturing and responding to individual student’s learning in real-time.

When considering the instructional needs of a fifth-grade student, this continuum provides a vast array of possibilities. The low-level option of the static webpage puts a considerable amount of control of the graphic design and content layout of the materials
in the writer’s hands, creating a scenario where much of the adaptation of materials is working with the individual student and responding to his/her needs. Since this option requires only the writing and updating of content there is little need to use client or server-side functions to generate content, allowing a classroom instructor to download these files to a storage device and access them without access to the Internet. This option would require a strong teacher-student relationship in order to develop and adapt the content to address the instructional individual needs of the student. This would create a Socratic method of instruction, one that is likely regarded as idealistic. When one considers the number of students educators have to work with on a daily basis, this option is quite impractical in both time and skill. A successful educator under this format would require a strong understanding of HTML coding in order to make changes to content in real-time to meet the student’s learning needs, and have mapped a sufficient number of instructional paths ahead of time to meet all of their students’ needs. Furthermore, when one considers the amount of interactivity within the material, this option might be better served with the instructor guiding an individual student or an entire class through the materials in a presentation style – prompting further guiding questions to keep students engaged.

The other option of utilizing a course/content management system appears to be a growing trend across the country and alludes to have some impact on student learning because of the accessibility beyond the classroom. These systems can provide numerous tools, including library access, student grades, communication devices, and anti-plagiarism software, giving prominence to the robust potential a system could provide to the entire learning experience. Many such systems require an existing server and an
annual licensing fee – creating an additional financial burden on school districts that many find difficult to fund. Furthermore, many of the built-in features to these systems are proprietary and are not easily adaptable to the individual learner. One programmer’s rationale to develop a tool within such a system may address functionality issues more than student learning.

Understanding this continuum and the limitations at each phase provides a better appreciation of why Orr developed EDU (public presentation, June, 2006). Working in an instructional support role, EDU was introduced as one of many online tools to increase overall work productivity. The use of EDU suggested that one could write a single question using one or more variables that the server could randomly assign a value – expanding that single problem to many variations based upon that variable. These possibilities indicated that more effort could focus on individual course development utilizing newer technology tools rather than expanding a single assessment.

At the server level, Java is the internal programming language used to manage the web delivery and randomization of variables. Java is very similar to other object-oriented programming languages that utilize objects, methods, and classes to define the organization and eventual operations. Utilizing JavaServer pages ("JavaServer Pages," 2008), Orr created QU code – a language through a web browser to write questions and processed by the server. To write the QU code, one can work in one of two fashions. The first is using a web interface displayed in Figure 5.1. In QU code, the topic is the uppermost object in which the user has control in generating questions.
Once a topic is created, one can begin to create questions within that topic through a WYSIWYG (what-you-see-is-what-you-get) environment. One can choose a variety of question types such as multiple choice, numeric, fill in the blank, or matching. For the multiple choice question format, the web environment allows the user to first type the sentence and then select the word(s) in which will the drop down menu will be create.

When Dr. Friesen first shared with me the materials he developed in EDU, he
began by illustrating the student’s interface – demonstrating how a question could be randomized. Figure 5.2 demonstrates a problem utilized in the three EDU sites focused on one of the later chapters. The objective focuses on dividing by powers of ten and then utilizes comments of moving the decimal based upon that number.

Figure 5.2 Random Problem and Comments in EDU.

Dr. Friesen spent much time emphasizing the connections to the grade book and
the commenting features – suggesting that there were ways to connect feedback to performance. The potential for immediacy with EDU suggested a larger hotbed of importance. Nevertheless, Dr. Friesen also emphasized the importance of being able to manually code the questions because some of the variables required additional mathematical computations. Figure 5.3 features the initial page of the programmed questions used throughout this study and Figure 5.4 illustrates the programming lines in the randomization process within one question.

Figure 5.3 Programming in QU Code.
Figure 5.4 Using the Algorithm/Randomization Process in EDU.

Appendix G shows the final version of the QU code used for all EDU sites in this project. Several points are noteworthy. The architecture of the language uses a top-down hierarchy. Each new line begins with qu indicating to the server a new line of code for one of the object, and each ends with an ampersand. The semicolon represents a line break where additional lines can be attached to each object before the end of the attributes.

There are already defined mathematical functions, such as greatest common denominator, modulus, and absolute value, suggesting similarities with other common programming languages. The algorithm property is where the randomization of variables
occurs – creating an opportunity to share that variable within both the comments and the question.

The biggest advantage using the QU code provides is that it allows the author to bring in other components – including HTML programming, access to outside resources on an additional server, and even more specific textual attributes.