Educational Software Evaluation Form: Towards a New Evaluation of Educational Software

Jason S. Wrench

Increasingly, the American classroom today is being revamped and turned into a unique structure where the teacher no longer says, “open up your textbooks and turn to page ten”; but rather, the teacher is now saying, “boot up your computer and open the happy trails program.” Since computers have come on the scene in education, more and more programs are released daily dealing with educational related issues. Mattas (1985) noted that there were nearly 8,000 different software programs available to teachers to choose from during the mid 1980s. Fifteen years later, that number has consistently increased (Maslowski & Visshcher, 1999).

In a recent study conducted by the Eisenhower National Clearinghouse at Ohio State University, the researchers examined 200 software packages intended for use in social studies and language arts instruction. Based on the evaluations, most of the software lacked important content. The researchers noted that the software was entertaining, but not instructional (USA Today Magazine, 1998). In an earlier study, the same researchers examined 175 math and science software programs and found the same results (BioScience, 1997). One of the major reasons for poor software in the schools deals with the evaluation of that software by education professionals. Despite the increasing number of software titles in education, the area of educational software evaluation has been increasingly more and more muddled because of a lack of consensus among software evaluators (Hardin & Patrick, 1998; Huber & Guise, 1995; Zane & Cornell). The purpose of this essay is to review relevant literature surrounding what makes educational software effective while purporting a new measure for examining educational software. The rest of this article will examine the history of computer software in the classroom, evaluate six areas that help make software effective, and demonstrate how the Educational Software Evaluation Form (ESEF) helps professionals evaluate software (Appendix A).

The history of educational innovation in the classroom is one that goes back to the beginning of time. Whether it was through the integration of speech or text, teachers have always used the innovations of the day to teach their pupils. When computer technology hit the United States and the world by storm, teachers were some of the first people to find unique ways to integrate the technology into their profession. Initially, students through the use of dummy terminals could access giant computer mainframes (Cosmann, 1996). These computer systems were seriously limited. One of the first giant leaps for educational software development occurred through a programming language called Logo. Logo was initially created in the 1960’s (Pfaffenberger, 1993). Logo enabled students to engage in active reasoning and problem-solving skills for the first time through the use of mediated technology. Logo was one of the most popular computer programs through the 1980’s. Most college students today still remember their first computer experience involved the Logo language through a graphics feature called “turtle”.

As computer systems gradually became smaller, giant mainframes went to the junk heap and are very rare in today’s computer world (Cosmann, 1996). Instead, the personal computer

Jason S. Wrench, earned is Bachelors of General Studies and Masters of Arts in Communication Studies at Texas Tech University. Mr. Wrench is currently pursuing a dual doctorate from West Virginia University in Curriculum and Instruction and Communication Studies.

has taken the world by storm. With the innovation of the personal computer, new educational software started to be developed in the 1980's (Mattas, 1985). The software itself has gone from the simple products created with Logo to the increasingly intricate designs that are seen today. Today, software exists for every discipline in the academy and at all educational levels with activities ranging from simple drills to advanced hypermedia-authoring tools (Cosmman, 1996).

With all of the software that exists today, it has become increasingly difficult for educators and curriculum experts to keep up with which software programs are the best. Since the 1980's, many different academicians have attempted to determine what to examine when deciding to purchase educational software and what makes software effective. The following discussion of the aspects of effective software will examine some of the early models used to examine educational related software. Every major researcher who has created a series of items for an individual to check off when evaluating software has created a new scheme. These new schemes have just confused the arena of software evaluation. All types of schemes have been created to evaluate software. Some of these schemes are very simple and only involve three major items to examine: solid instructional design, flexibility, and results (WD&S, 2000). Other schemes have integrated up to eight different components: accuracy, learning style, user experience, curriculum, pedagogical philosophy, learning objectives, format, and instructor preferences (Hardin & Patrick, 1998).

With the myriad of different possibilities for areas that should be evaluated, it is easier to see where software evaluators agree on what is effective software than where they disagree. Six basic areas can be evaluated to see what software evaluators believe are essential to effective educational software: curriculum content, instructor use, student use, program content, program operation, and publisher information. The first area that needs to be examined deals with the concept of curriculum content. When looking at various researchers opinions of what should be examined by software evaluators, most of them listed the curriculum as being one of the top aspects to be examined (Huber & Guise, 1995; WD&S, 2000; Komoski, 1995; Kokol, 1991). One component that makes educational software effective is when it is useful and collaborates with actual curriculum. Too often teachers just use technology for technology's sake without having any specific curriculum goal (McComb, 1994). The primary goal that Perkins (1988) believes that all educational software should have is generalization. Generalization is the ability to generalize what is learned on the computer to contexts outside the computer. In a study conducted by Howe (1989) noted in Zane and Frazer's (1992) article, it was found that many children who mastered spelling vocabulary words on the computer were unable to define and spell those same words in other situations. Effective software would have built in measures that can prevent this from occurring. One way for software manufacturers to make sure that generalization occurs is to provide teachers with activities off of the computers that help them with the generalizability of the learned information.

A second aspect that needs to be examined when looking at the curriculum is whether the software has clear objectives. A singular piece of software cannot save the world, but can be used for a specific application. For a piece of software to be effective, the programmers needed to have focused in on a specific goal (Maslowski & Visscher, 1999). At the same time, educators using software should not expect more from the software than is realistically possible. A software designed to help increase spelling effectiveness should not be expected to then teach students mathematics. Also, when making sure that a piece of software would be effective for a classroom, evaluators need to make sure that the software is age appropriate yet allows the learner to be challenged consistently as he or she increases her or his use of the software.

The second major aspect to examine when determining if educational software is effective is to examine it from the instructor's perspective. For a software to be used effectively, a teacher must realize that he or she has a definite role to play with the software's implementation. Too often teachers who use technology in the classroom do so haphazardly seeing the computer as just a glorified toy to occupy their student's time. One way for software to be used effectively is to provide teachers with suggestions on how to use and integrate the software into already existing curriculum (Bitter & Camuse, 1988). Another way to make the software more effective for the teachers is to provide them with a master explanation of the software and its various parts. Komoski (1995) believes that teachers who will be using software should be actively involved in the search for software that they will be able to use the most effectively in their classrooms.

The third major consideration that needs to be examined deals with how a student will use the software. For software to be truly effective, it must take the user into account. In the case of educational software, understanding the student is essential to making software effective. Hardin and Patrick (1998) mention that for educational software to be effective it must take into account different learning styles. Mirsha and Young Zhao (1999) mention that first and foremost software developers must keep human concerns at the center of all effective software development. Another major consideration when thinking about the student, is whether the student will be able to generalize the information to contexts away from the computer environment (Perkins, 1988). If a student is unable to generalize the information, this means that something has gone wrong with the process of the software creation itself.

Another major consideration for effective software today is making sure that the software is multicultural and gender sensitive. Consistently, studies have shown that women, though purportedly equal, still make less money for equal work when compared with men (Kilbourne, 1999). Unfortunately, these gender inequities have often made their ways into the classrooms as well (Matthews, Binkley, Crisp, & Gregg, 1997-98).

One example of the impact gender has on software was seen by Yelland (1996) who wanted to examine how males and females collaborated while using computer technology. Yelland used 60 children in the second grade going to school in a state primary school. The students were broken into collaborative pairs (male-male, female-female, and female-male) for the duration of the project. The researcher found that female-female groups offered more information and explanations to each other, asked for more proposals from each other, offered more proposals from each other, and were more likely to agree with the proposals offered than either the male-male or female-male groups. In essence, female-female collaborative teams tended to be more problem solving oriented from an equitable stance than the other dyads. In a similar study conducted by Inkpen, Booth, Klawe, and Upitis (1996), the researchers found that when males and females would work together to solve computer (educational game) problems, the pair could solve more problems than either one of them could alone. At the same time, girls tended to solve significantly fewer problems than boys in general. (Actual statistics not shown in the article).

One of the first ways to make distance education collaboration work is to make it first and foremost egalitarian. As was noted by Little-Reynolds (2000), a large quantity of the software that educators are currently trying to use in their classrooms is simply not multicultural or gender sensitive. One possible explanation for the results found in the Inkpen, Booth, Klawe, and Upitis (1996) study could pertain to whether the software used in the study was created from a masculine standpoint. If the software was created from a male oriented perspective, then the results from their study could be skewed towards males solving more problems than the females. Evaluative techniques need to be developed to clarify how an individual sets out to evaluate educational software. In essence, for software to truly be effective it needs to be equitable for all users.

The fourth and fifth concepts that help make a piece of software effective concerns the software from a software developer's perspective. Software developers need to make sure that the software components are first pleasant and second error free and easy to use. Mirsha and Yong-Zhao (1999) note that software should be aesthetically pleasing to the people who are going to be using the software itself. Smith and Tabor (1996) discussed aesthetics when they wrote:

> There is a delight in a program that is rigorously consistent, elegantly clear and lean, where sound and vision are perfectly at one, or where the representation chosen neatly fits the ways that users think about what they are doing. These qualities cannot be added on at the end: they are integral to the design engineering of the product. (p. 42)

The overall quality of the software must be aesthetically pleasing for the user. Everything that a user sees on the screen, or does not see, can affect the way that he or she uses the software. It is for this reason that extensive testing with software is a must before it is released to the public. As Henson (1991) wrote, effective software is one that is built on objectives, created on a course model, then evaluated, and then modified as many times as necessary.

A second major concern from the perspective of the software developer that will determine the effectiveness of the software is the software's usability. Usability is usually referred to as the degree to which software assists a user in completing necessary tasks (Levi & Conrad, 1998). As was noted by Ahern (1996), the way a computer screen is laid out will determine whether or not a user finds it effective. One possible way for software evaluators to determine whether their product is as effective as it should be is to follow the process discussed by Nelson, Bueno, and Huffstutler (1999). Nelson et al. (1999) watched participant's patterns of activities, matrices were then created tracking the user's navigation of the software, and then the matrices were analyzed using a Pathfinder algorithm to see whether the software was user friendly.

Beyond looking at the aesthetics and usability of the product, software developers and evaluators should look for the simple things that most individuals overlook when evaluating a program. Bitter and Camuse (1988) noted that some items to look at are the system space needed, computer compatibilities, online instructions, and other system related issues. Along with these is the notion of portability discussed by Bengtsson (1990). Portability looks at whether a piece of software will become obsolete when new computers come out. In the past, a lot of software has been discarded simply because it was not compatible with newer systems when they were adopted by schools. Anyone buying software should not just look for software for tomorrow, but rather search for software that will last beyond tomorrow.

A final major component that makes software effective is technology support and verification from the publisher. When looking at software, find out what kind of options the publisher of that software will provide individuals once the product is bought. As Henson (1991) noted, a good software publisher is willing to work with people who buys their products because it allows the publisher to see where the bugs in the program are and correct them when new upgrades become available.

Unfortunately, there are more software programs out there than there are good software publishers. As was noted by BioScience (1997) and USA Today Magazine (1998) a lot of the software packages out on the market today are not measuring up to the promises that they make. Many software evaluators mention that one of the critical aspects of software evaluation is determining whether the software is effective and produces results (WD&S, 2000; Hardin & Patrick, 1998; Zane & Frazer, 1992). In a study conducted by Zane and Frazer (1992), the researchers attempted to contact thirty-four different corporations developing and marketing educational software. These companies distributed 95 different educational programs that covered a wide range of academic content. The researchers asked the corporations to supply an evaluative data, research findings, or field-test results concerning five basic areas: time for students to learn and operate the software; time required for students to complete all lessons contained in the program; documentation that students learn skills, knowledge, or fluency from the software; comparative data indicating that the software has an advantage over other independent means of instruction; and documentation that students who learn from the software perform better on traditional methods of evaluation for the same subject matter, or any other evidence of generalization of learned skills from the computer to other contexts. A total of 15 (44%) of the companies contacted responded. These companies accounted for 49% (47) of the 95 pieces of software evaluated. Not one of the companies was able to provide any documented support showing their software's effectiveness.

Now that a look at the literature surrounding what makes effective software has been conducted, an explanation of the Educational Software Evaluation Form (ESEF) can occur. The ESEF was created to take into account a number of ideas that arose in the literature. The basis of the ESEF comes from a form created by Bitter and Camuse (1988). The original form created by Bitter and Camuse examined program content, program operation and documentation, instructor use, and student use. Though these items were important, it became obvious that these were not the only areas that people who are evaluating software should be concerned. For this reason, information on training, curriculum content, and publisher information was added to the measure. In addition, two items were added to the student use section: "Allows for students to cooperate with other students to complete the activities" and "This software allows students with different learning styles to be equally effective." The ESEF is found in its entirety in Appendix A.

This article has examined the components that come together to make effective educational software. Though this article only began to scratch the surface of what constitutes effective software, this article hopefully starts a conversation about what is and is not effective software and software evaluation. Though the Educational Software Evaluation Form is not a perfect instrument for evaluating software, hopefully it can be a more beneficial form than current ones in the field. With the more research that is generated in the field of software evaluation, the better grasp software adopters can stand firm in their choices of educational technology. And as the consumers of educational software become more shrew and know what to look for, maybe the findings seen in the Ohio studies or the Zane and Frazer's (1992) findings will no longer occur. And hopefully the next time a teacher has her or his students boot up their computer and open that new software package, he or she will know that her or his students are getting the best educational opportunity possible.

References

Ahern, T. C. (1996). Effect of window state on user behavior in an on-line computer mediated conference. Conference on Human Factors in Computing Systems. Van Cover, British Columbia, Canada, April 12-18. [On-line] Available at: http:www-csc195.Indiana.edu/csc195.ahern.html.

BioScience. (1997). Educational software gets failing grade. Bioscience, 47, 720.

Bitter, G. G., & Camuse, R. A. (1988). Using a microcomputer in the classroom (2nd Ed.). Englewood Cliffs, New Jersey: Prentice Hall.

Cosmann, R. (1996). The evolution of educational computer software. Education, 116, 619-623.

Hardin, L., Patrick, T. B. (1998). Content review of medical educational software assessments. Medical Teacher, 20, 207-212.

Henson, K. L. (1991). The use of prototyping for educational software development. Journal of Research on Computing in Education, 24, 230-240.

Howe, J. A. (1988). Assessment of skill generalization taught by computer software. Unpublished Maser's Thesis, Russell Sage College.

Huber, J. T., & Guise, N. B. (1995). Educational software evaluation process. Journal of the American Medical Informatics Association, 2, 259-96.

Inkpen, K., Booth, K. S., Klawe, M., & Upitis, R. (1996). Playing together beats playing apart: Especially for girls. Conference paper presented: Conference on Human Factors in Computing. Vancouver, British Columbia, Canada. April 13-18. [On-line] Available at: http://www-csc195.indiana.edu/csc195/inkpen.html.

Kilbourne, J. (1999). Deadly persuasion: Why women and girls must fight the addictive power of advertising. New York: The Free Press.

Kokol, P. (1991). A new microcomputer software system evaluation paradigm. Journal of Medical Systems, 15, 269-96.

Komoski, K. (1995). Seven steps to responsible software selection. Educational Resources Information Center Digest, May, 3-4.

Little-Reynolds, L. (2000). Evaluating software to address multicultural issues: An exploratory investigation. Conference paper presented: EERA. Clearwater, FL. February 17.

Levi, M. D., & Conrad, F. G. (1998). Usability testing of World Wide Web sites. Washington, D.C.: Bureau of Labor Statistics. [On-line] Available at: http://stats.bls.gov/ore/htm_papers/st960150.html.

Matthews, C. E., Binkley, W., Crisp, A., & Gregg, K. (1997-1998). Challenging gender bias in the fifth grade. Educational Leadership, 55 (4), 54-57.

Mattas, L. L. (1985). Only the best: The discriminating software guide for preschool-grade 12. Sacramento, CA: Educators News Service.

Maslowski, R., & Visshcher, R. (1999). Formative evaluation in educational computing research and development. Journal of Research on Computing in Education, 32, 239-256.

McComb, M. (1994). Benefits of computer-mediated communication in college courses. Communication Education, 43, 159-170.

Mirsha, R. Young Zhao, P. (1999). From concept to software: Developing a framework for understanding the process of software design. Journal of Research on Computing in Education, 32, 220-239.

Nelson, W. A., Bueno, K. A., & Huffstutler, S. (1999). If you build it, they will come. But will they use it? Journal of Research on Computing in Education, 32, 270-387.

Pfaffenberger, B. (1993). Que's computer user's dictionary (4th Ed.). Indianapolis: Que.

Smith, G. C., & Tabor, P. (1996). The role of the artist-designer. In T. Winograd, J. Bennett, L. DeYoung, & B. Hartfield (Eds.) Bringing design to software (pp. 37-57). New York: Addison-Wesley Publishing.

USA Today Magazine. (1998, April). Educational software gets low marks. USA Today Magazine, 126 (2635), 15.

WD&S. (2000). How to evaluate educational software. Curriculum Review, 39, 15.

Zane, T., & Frazer, C. G. (1992). The extent to which software developers validate their claims. Journal of Research on Computing in Education, 24, 410-420.

Appendix A

Educational Software Evaluation Form (ESEF)

Title:	Grade Level:
Subject Area:	
Memory:	Free Hard Disk Space:
CPU:	
Operating System:	
Additional Hardware:	
Additional Software Required:	
Publisher:	
Publisher's Address:	
Phone:	Fax:
Web Support:	
Storage Medium:	
Cost:	

Training Provided:	YES	NO
Support Provided:	YES	NO
WWW Support:	YES	NO
Phone Support:	YES	NO
In-house Support:	YES	NO

Brief Description of the Program:

The following is a series of questions that can help you determine whether this piece of software is beneficial for your classroom. The first thing to do is to examine each question and determine whether it applies to you. In the "weight" column record whether the question applies to your needs or not (0-"does not apply to my needs" to 5-"strongly applies to my needs"). After you have determined the weights go back through and then evaluate the software by ranking the software on each statement in the "rating" column (0-"this softwa re does not do this" to 5 "this software does this very well"). Then multiply the rating by the weight to come up with a total. After each section add your totals. More instructions will follow after you have filled out the entire evaluation form.

Characteristics: Curriculum Content

	Ratin g (0- 5)	*Weight (0-5)*	*Total*
1. This software would work with existing curriculum			
2. Breadth of the content in the software			
3. Depth of the content in the software			
4. Software has clear learning objective s			
5. Content can advance as the user (learner) advances			
6. Age appropriate			
7. Shows a diverse group of people (women, men, different ethnicities, cultures, etc. . . .)			
Total Program Operation/Documentation Score			

Characteristics: Instructor Use

	Ratin g (0-5)	*Weight (0-5)*	*Total*
1. Instructional objectives are clearly stated			
2. Suggestions for curriculum integration			
3. Prerequisite skills for students indicated			
4. No need for instructor to assist users			
5. Useful teacher manual included with the software			
6. Suggested grouping arrangements			
7. Useful student workbook provided			
8. Useful ditto masters provided by the software publisher			
9. Interesting follow-up activities and/or projects suggested			
Total Program Operation/Documentation Score			

Characteristics: Student Use

	Ratin g (0-5)	*Weight (0-5)*	*Total*
1. Requires no computer knowledge			
2. Does not require the student to reference manuals			
3. High student involvement			
4. Provides students with a summary of performance			
5. Shows no racial, sexual discrimination			
6. Encourage cooperation			
7. Student control over rate of presentation when appropriate			
8. Length (time) of simulation is appropriate			

9. Allows for students to cooperate with other students to complete the activities			
10. This software allows students with different learning styles to be equally effective			
Total Program Operation/Documentation Score			

Characteristics: Program Content

	Rating (0-5)	*Weight (0-5)*	*Total*
1. Realistic simulation of events			
2. Valid formulas used to compute outcome			
3. Accurate content			
4. Appropriate use of color			
5. Appropriate use of graphics, video, or animation			
6. Appropriate use of sound			
7. Accomplishes stated objectives			
Total Program Operation/Documentation Score			

Characteristics: Program Operation and Documentation

	Rating (0-5)	*Weight (0-5)*	*Total*
1. Allows user to correct typing errors			
2. Documentation available and clearly written			
3. Clear, nicely formatted screen displays			
4. Incorrect use of keys or commands does not cause the program to abort			
5. Menus and other features make the program “user friendly”			

6. Instructions can be skipped if already known			
7. Uses correct grammar, spelling, and punctuation			
8. Clear and useful summary of program operation provided			
9. Loading instructions are clear; program is easy to load			
10. Bug free; program runs properly			
11. Uses computer capabilities well			
12. Accepts abbreviations for common responses (i.e., Y for YES)			
13. Operation of program does not require user to turn computer on and off			
14. Readability of text is appropriate for intended user			
Total Program Operation/Documentation Score			

Publisher Information

	Rating (0-5)	*Weight (0-5)*	*Total*
1. Publisher used appropriate theoretical basis for software development			
2. Publisher has research demonstrating the effectiveness of the software			
3. Publisher is considered a reputable company in your field			
4. Publisher is willing to provide information about satisfied customers for you to contact about the software			
5. Publisher is willing to discuss why some customers have been dissatisfied with the product			
6. Publisher solicits teacher recommendations for future versions of the software			
7. Research not directly conducted by the publisher has shown			

this product to be effective			
8. Reviews of this product have been very positive			
Total Program Operation/Documentation Score			

Evaluation Summary: *(Main Strengths and Weaknesses)*

Strengths

Weaknesses

Total Points for all Characteristics: ________
(add all totals together)

Total Points Possible: ________
(Add all weights and multiply by 5 to determine total possible points)

Percent Rating: ________ percent

How to interpret results:

90 – 100%	This would be a good software decision
80 - 89%	This piece of software is beneficial, but may have some serious reservations
70 - 79%	This piece of software is probably not going to get the results desired
60 - 69%	This piece of software has too many reservations to adopt in its current state
Below 60%	This piece of software is not tested and not beneficial