DESIGNING INTERACTIVE INSTRUCTIONAL TOOLS TO SERVE BOTH STUDENTS AND INSTRUCTORS - A CASE HISTORY

Henderson, R. C[1]., Lowhorn, D. E.[2], Tindall, W. R.[3], Larimore, D. L.[4]
[1]Dept. of Civil & Environmental Engineering, Tennessee Technological University
[2]Educational Technologies Center, Tennessee Technological University
[3]Dept. of Computer Science, Tennessee Technological University
[4]Dept. of Curriculum & Instruction, Tennessee Technological University

Abstract

The Homework Laboratory® (The HWL) is a CD-based educational tool for use in fundamental science and engineering courses. Capable of being used with virtually any quantitative course of study, it is intended to help students learn the course material in a more effective manner and to make the administration and presentation of the course easier for the instructor. The National Science Foundation sponsored the testing of the software (using control and experimental groups of students) to assess its effectiveness at improving student understanding. The test program was conducted over a three-year period in engineering mechanics classes at The University of Texas at Austin (UT) and Tennessee Technological University (TTU). Also, in order to assess the modularity of the program (that is the ease with which new courses may be implemented) the software was modified and implemented in a high school statistics course.

This paper explores the technical design approaches, implementation, and lessons learned throughout the three-year development and modification of The HWL.

Introduction

Software researchers have identified numerous discrete phases inherent in software development.[9] These include the following:

1. Problem Definition
2. Requirements Analysis
3. Architecture Design
4. Detailed Design
5. Coding and Debugging
6. Unit Testing
7. System Testing
8. Corrective Maintenance
9. Functional Enhancements

Each of these phases is, to a large degree, cyclic and interdependent and must be revisited throughout the life of the software in order to maintain usefulness and efficiency. This paper discusses the development of an educational software instrument entitled *The Homework Laboratory*® (The HWL), and the above development phases will be used as a framework for discussion. Items 4,5, and 6 represent what is commonly called software construction, where much of the development effort is focused; thus, these elements will be emphasized herein.

Problem Definition

One of the primary means for an instructor to help a student master a fundamental science or engineering topic is to require the completion of representative homework problems. Core courses are often homework intensive, and for the student to achieve the most benefit from the homework process, the instructor must spend a significant amount of time grading and correcting assigned problems. Despite this effort, however, most instructors have observed students with near-perfect homework scores who perform very poorly on tests. Since testing concepts are generally based on the principles learned by completing homework, one must conclude that the student did not really understand these principles and/or was unable to apply them under time constraints. This phenomenon happens frequently and is generally an indicator that the student relied

heavily on other sources (group study, homework files, etc.) in attempting to learn the material.

The objective of The HWL was to reduce this lack of preparedness on the part of the student (and the associated deficiency in real learning) while at the same time save the instructor a considerable amount of time normally associated with grading and coaching. To determine the effectiveness of this learning instrument, the National Science Foundation (NSF) funded a three-year study comparing use of The HWL with the traditional homework model. This was accomplished using experimental and control groups of students at The University of Texas at Austin (UT), Tennessee Technological University (TTU), and Monterey High School (MHS).

Requirements Analysis

Before initiating development of The HWL, a literature review indicated that computer-based instruction had been proven to be effective at enhancing student comprehension and retention.[8] Likewise, over the last decade, budgetary constraints and improvements in multimedia technology have been pushing universities toward more reliance on technological approaches.[6,4,11] Furthermore, computerized apprenticeship methods for learning, emphasizing practice and often involving commercial software and other tools, have been shown helpful for students in fundamental engineering and/or science courses.[10,7]

However, many of the computerized methods, helping students learn concepts by simulation, parametric examination, audio/ video coaching, etc., were often found to have two primary disadvantages:

(1) they were unrelated to the student's grade and, therefore, were frequently viewed as another expense for which the student had little need, and

(2) the instructor often perceived them to be peripheral material with little added benefit over text and classroom explanations.

Since both students and instructors are heavily involved in using any computer-based approach, the learning instrument must provide benefits for both audiences. Therefore, to accomplish the objective of improved student learning and a reduction in course responsibilities for the instructor, the requirements analysis indicated that a computer-based learning instrument must accomplish the following specific functions:

(1) randomize the variables for a student's homework so that each problem is different for every student;
(2) score each homework problem;
(3) instruct the student as necessary regarding calculation errors;
(4) keep track of individual and class homework scores for the instructor;
(5) create randomized and timed practice tests for the student; and
(6) provide a virtual classroom to assist the student with general concepts before attempting homework or a test.

Architecture Design

The six specific elements listed above were accomplished using an instructor version (whereby the instructor can assign homework, track grades, etc.) and a student version (that grades homework, coaches the student, tests the student, and teaches the student in the virtual classroom as shown on the main screen of Figure 1). In addition, it was determined that the overall software must be modular in nature (i.e., designed with a clear separation between navigational and computational functions so that new subject areas may be easily implemented). The following components of the software architecture were crucial to achieving these design criteria:

(1) the development environment;
(2) the database;

(3) the Student Data File (SDF);
(4) the main navigation and operation code (NavOp Code); and
(5) the problem code.

Development Environment: Several different programming languages/environments were considered. The original prototype of the program was developed using Macromedia's Authorware®. While Authorware® provided for good interface development, it did not provide sufficient power or calculation tools for the necessary mathematical programming. Visual Basic was also considered, but was decided against because of the amount of code that it needed to perform certain operations. Finally, programming in C was investigated. However, at the time, the graphical portion of the programming in C was significantly more labor intensive than for either Visual Basic or Delphi. Delphi[2] was eventually chosen as a mix between low-level code control and high-level, Rapid Application Development (RAD) tools.

Delphi's RAD interface takes input from the programmer in two primary ways: through graphical forms and through code units. Forms in Delphi act as a canvas on which the programmer designs the layout of the graphical user interface (GUI). Each of these forms then has a unit associated with it, and it is in this unit that the code is written that drives the actions of the form. The forms also function as a container for components (e.g., timers, list boxes, etc.) that respond to user input. Figure 2 shows the main form used by the HWL. Many of the operations of the program are initiated by code associated with this form. For example, the main screen shown in Figure 1 is one configuration of the main form (though only a small portion of the components of Figure 2 are used for the main screen). Likewise, the Tgauge and Ttimer shown in Figure 2 are Delphi components that are used within the HWL to monitor student progress on tests and homework as well as to provide messages and initiate administrative functions periodically. The combination of RAD tools and Object Pascal in Delphi was found to simplify the development of the user interface and provide the speed, efficient memory usage, and calculation power necessary for The HWL project.

Database: Though The HWL can be used with any text or quantitative course of study, *Engineering Mechanics – Statics*[1] was chosen for the research. The text contains 1502 problems and all were programmed into the software. [The instructor, using the instructor version, decides which of these will be homework problems and which will be available for practice.] Each problem consists of a unique set of data that includes the problem number, image name (if applicable), the problem text, all of the information necessary to properly display the problem, and the problem code. This information can become voluminous and much of it must be displayed graphically as shown in Figure 3.

Therefore, a series of Paradox database tables (managed using Borland's Database Desktop) are used to store all of this information for each problem except for the problem code. The Borland Database Engine (BDE) is then used to connect the program with the database tables. Each table in the database is associated with one chapter in The HWL. By doing this, related problems are grouped together. This makes it easier during the development stage by allowing one person to work on one chapter while another works simultaneously on another.

When a problem is executed, the information needed to correctly draw the screen is retrieved from that problem's entry in the corresponding database table. The key values in the database tables are the concatenation of the needed data to uniquely define a given problem. The key takes the format of [Chapter] [Section] [Problem]. For example, the information for problem number 16 in Chapter 3, Section 3, is retrieved from the database as 33016 in the Delphi code.

The primary advantage of the database approach is the isolation of problem data and coding from the main navigation and operation

code. In this way, a new course may be implemented by extracting the old database information (and associated code) and inserting that for the new text.

Student Data File: The HWL also uses and creates data associated with the current user. This data is stored in the current user's Student Data File (SDF). The user must supply an SDF and the password for that file before accessing any of the features of the program. This file contains the following information: (1) the user's name and password; (2) a record of how much time the user has spent in each section of The HWL; (3) which problems are assigned as homework problems for the course; (4) seed values for random number generation for problems that the user has started; (5) grades for homework problems the user has completed; and (6) a record of practice problems the user has completed or started and grades for the completed ones.

This file is encrypted and is updated throughout the use of the program, not just when the user closes the application. A generic SDF is created by the instructor (using the instructor version) at the start of a semester and distributed to each student. The HWL modifies this SDF as the student uses it throughout the semester. Encryption of SDFs will be discussed in more depth later in this paper.

Main Navigation and Operation Code: The main navigation and operation code, or NavOp Code, is where all GUI design, navigation operations, and data handling/processing functionalities are implemented. This body of code manages the GUI, implements program response to user actions, accesses the database for problem data, handles retrievals and updates involving the current SDF, and calls problem procedures. The NavOp code is stored in various units, most of which are associated directly with a specific form. Each of these units of code handles interaction with the user when the user is viewing its associated form. There are also a few units of code that are not directly associated with a form, but instead contain global procedures and variables that are used at various stages during the flow of the program's execution.

Problem Code: The calling of a problem procedure is where the NavOp code transfers responsibility to the final division of The HWL's software architecture, the problem code. In Delphi, forms are the design palette for the GUI that a program uses, while units are where the code is stored that drives those forms. However, a program can also have units that are not directly associated with a specific form. These merely contain code. That is how the problem code is stored. There is an individual unit for each chapter that contains procedures associated with a given problem. These procedures perform the randomization of the problem variables and the actual mathematical calculations necessary to work the problem. These answers are then passed back to the NavOp code to be compared to the user's proposed solution(s).

Figure 4 shows the code for problem 33016 (i.e., Chapter 3, Section 3, Problem 16; displayed graphically in Figure 3). This problem has three variables (W1, W2, and theta) and a one-part solution (F). For any problem, up to six randomized input values and six solution values may be required. In Delphi, all variables for a procedure must be declared at the beginning of the procedure in the var section (Lines 2-5). Then, the main body of the procedural code starts (Lines 5-17). The first task this procedure performs is to randomize the variables and display them on the screen. The displaying of the randomized variables by the problem procedures is the only overlapping of the main NavOp code and the problem code divisions.

As shown on Line 10, not all of the variables are randomized. In general, no more than 3 variables are randomized on a given problem. [Randomization of multiple variables can lead to numerical solutions that, though mathematically correct, are physically impractical or impossible. The intent is to

provide enough unique solutions to prevent duplication, while avoiding the problems associated with broad randomization. It was found that for engineering mechanics, generally, 1000 unique problems are easily achievable by randomizing only two variables.]

After the variables are set, the solution is calculated (Line 13). Then, the solution, F, is passed out to the Answer array, which is used by the NavOp Code to determine if the user's input solution is correct. Though there are numerous problems in a text, the coding for each problem is generally short – usually taking a qualified student programmer about 20 to 30 minutes to complete.

Construction

Software construction generally includes elements of detailed design, coding and debugging, and unit testing. As these phases were cycled through, the following three main design goals were the focus of the construction process for The HWL:

(1) simple and intuitive navigation;
(2) appealing and appropriate GUI; and
(3) emphasis on function and order.

Designing an interface that was easy to use was of paramount importance. Typical science and engineering courses inherently involve a great deal of work for students (i.e., attending class, reading the text, studying lecture notes, working homework, etc.) as well as for instructors. Thus, learning a new software package detracts from the primary focus – the course material. Therefore, straightforward access to the major segments of the program (Homework, Practice, Classroom and Test; see Figure 1) was the driving force in the interface layout. A significant amount of construction time and effort was also invested in, making the program similar (in terms of graphics and layout) to the text it was accompanying. The intent was for the students to view the software as a seamless extension of the required text, not supplementary software. Inside all of that, a conscious push was made toward making the interface as minimal as function would allow. The GUI needed to be easy to use and intuitive, but it did not need any unnecessary options/functionalities that would only clutter the interface.

While designing The HWL with these three major goals in mind, several smaller, more specific areas of software construction took on major importance. These areas, listed below, were the principal areas of effort and planning for The HWL and will be discussed in this section.

- Thumbnail Images and Problem Selection
- Classroom Topical Search
- Encryption of the SDF
- Integration of PowerPoint and Excel
- Button Highlighting

Thumbnail Images and Problem Selection: Thumbnail images on the problem selection screen (as shown in Figure 5) were designed and implemented as a response to each of the design goals. When a student highlights a section of a chapter from which to work a problem, a box appears that lists all of the available problems from that section. A thumbnail of the image associated with it accompanies each problem. In this way, the student sees immediately all of the problems in that section that have been assigned by the instructor (all other problems in the section are available in the practice portion of The HWL).

Furthermore, the thumbnail graphic (which is a smaller version of the actual problem graphic) helps the student to determine the type of problem that has been assigned (without the necessity of beginning the solution process). The construction of this tool was completed using an instance of the Object Pascal TlistBox component. As the cursor moves over the chapter section, a subroutine displays the TlistBox, which has been previously filled using the AddObject method. The AddObject method allows for a string of text as well as a Tobject to

be associated together as one entry in the TlistBox.

Classroom Topical Search: The classroom section is designed to help users learn about topics covered in the current textbook. It does this by allowing the user to select a section to review or to search for a topic to review. It then lists the related sections. If the user wishes to select a section to review, then the sections are simply listed in the "By Section" option (Figure 6). However, if they wish to search the text for a given topic and see all related sections, they can use the "By Keyword" option. Searching by Keyword is available because many times users want to review a topic, not necessarily a section.

To make the search fast and easily understood, an input box and a result box are shown. The searching is setup so that when the keyword input value changes (even by only one character), the result box is updated automatically. This way, when the user finishes typing in the desired keyword, the results of that search are already finished and are being displayed.

This method of searching and displaying the results is accomplished using a second thread. While the main process thread continues to handle the main interactions with the form, the second thread is setup to react to any changes to the keyword input box. By using a second thread, the searching is done virtually simultaneously with the input.

Encryption of the SDF: With The HWL, as with most software packages involving individual user data, security of that data is very important. For educational software involving grades and network availability to students, it is just as important to keep the user from unauthorized access to their data as it is to keep other users from accessing it. Therefore, the SDF must be encrypted to secure passwords and to prohibit the viewing of grades by anyone except the instructor.

The SDF is a text file that is encrypted at the bit level, with each string in the file being encrypted/decrypted based on a different random key. It is a symmetric encryption algorithm, which means that the same keys are used to encrypt the data as are used to decrypt the data.

Integration of PowerPoint and Excel: The classroom section uses an external program, namely Microsoft's PowerPoint Viewer. The final goal of the classroom section, whether the user directly selects a section or searches for a topic, is to provide a way for the user to review the desired subject matter. The actual review is displayed as a PowerPoint Presentation using Microsoft's PowerPoint Viewer program (Figure 7). (The Microsoft PowerPoint Viewer is included on The HWL CD-ROM, and, if needed, is installed when The HWL is installed.) The presentations are designed to be visually similar to the text as well as other sections of The HWL and to provide a sense of continuity to the user.

When the user selects a section, the PowerPoint Viewer is started with the corresponding presentation passed in as a parameter. This is accomplished using the CreateOLEObject procedure, which takes as a parameter the CLSID for the object to be created. The CLSID is the value that represents the program in the Windows Registry. The CreateOLEObject procedure takes the CLSID, accesses the registry, and then creates an object of the associated class. This object is then started using its newShow method with the filename of the appropriate PowerPoint presentation passed in.

The Instructor Version of The HWL also uses an external program to open SDFs and show the instructor the grades for the student. The instructor may also open multiple SDFs at once. After doing this, they have several options:

1. save the information for the opened files in a text file or a comma-separated file (CSV File);

2. copy the information to the Windows clipboard; or
3. start Microsoft Excel with the opened information passed via a CSV File (if Excel is installed).

With the third option, the instructor has all of the tools associated with Microsoft Excel with which to manipulate the data.

The method used to start Microsoft Excel is different from the method used to start the PowerPoint Viewer. Delphi comes packaged with some helpful file-interaction procedures contained in the FMXUtils unit. One of these procedures is ExecuteFile. This procedure acts as a wrapper for the Windows API function, ShellExecute. While ShellExecute requires several obscure parameters, the ExecuteFile procedure only requires the name of the file to execute (in this case, 'excel.exe'), the name of a file to pass in, a default directory to look in for the file, and the required method for window display.

The two different approaches for launching external programs were implemented at different stages during the development process. After comparing the two, the first method (i.e., OLE procedure) is preferred due its simplicity in interacting with the external program.

Button Highlighting: The major method of interacting with The HWL is through buttons. There are buttons on almost every screen that provide the user with clearly defined options for navigating through the program. These buttons are semi-transparent and are highlighted when the cursor is over them.

This is accomplished by using several instances of the Timage component. One is called the mousefield and the others are called hotspots. The mousefield is set to cover the entire screen, while the hotspots are aligned with the button images that are drawn as part of the background. Whenever the user moves the pointer on the form, the mousefield's onMouseMove procedure is called. This procedure, using several other subroutines, checks to see if the pointer is currently over one of the active hotspots. If it is, that hotspot is highlighted and all others are de-highlighted. If the mouse is not over a hotspot, then any previously highlighted hotspot is de-highlighted.

The development of The HWL also involved numerous cycles of coding and debugging. For most of the major elements of the program, test applications were written to test each discrete element. For example, the TListbox for section selection (described above) was first developed as a separate application. After the debugging was complete on the test unit, it was implemented in The HWL. Most of the major elements of The HWL were coded, debugged, and integrated in this way.

System Testing

System testing of the program involved both in-house testing by the development team and external testing by three different institutions (UT, TTU, and MHS). The HWL software was developed for the Windows operating system, and students accessed the software by campus network or loading the software onto individual PCs via CD. The in-house tests involved numerous cycles of checking all of the options of The HWL (both on the network and on individual machines) for expected performance and changes to the code to fix any bugs found. These formal tests were performed by lead programmers on the project, whose goals were specifically to try to find faults in the program. Other project personnel conducted informal tests as they used the program to check the problem coding and the database for errors. [Approximately 30 people contributed in various capacities to the software development.] As they encountered problems during their use of the program, they were reported and remedied.

During external testing, instructors and students at UT, TTU, and MHS used the program and reported both problems and suggestions throughout each semester via

formal and informal surveys. [Approximately 800 students used the software over both a large campus network (UT) and a smaller campus network (TTU), as well as individual machines.] The development team then investigated these problems and suggestions and, when appropriate, changes were made. The input from these impartial instructors and students also provided valuable feedback on the effectiveness of The HWL as a learning tool.[5]

Corrective Maintenance

Data gathered from the internal and external testing prompted changes to The HWL in terms of both corrective maintenance and functional enhancements. Early in the development process, most of the corrective maintenance issues were low-level in nature and often involved individual problem code or the appropriate functioning of utilities such as printing, saving files, or navigation. These types of corrections were made frequently throughout the software development and implementation cycles in response to suggestions made by students and instructors each semester.

A larger maintenance issue involved the installation of the program. The changes in configuration from one PC to another required an installation script that was robust enough to perform correctly under a wide array of hardware, software, and network configurations. The installation script was originally developed using InstallShield Express for Delphi. However, as the size of The HWL grew to over 6,500 files, a more powerful and flexible installation protocol was required. Thus, ultimately, the full version of Install Shield was used on the TTU and UT campuses. Final versions of the installation script included provisions for several different installation settings. Traditional, Full, and Compact installations were made available, and options regarding implementation on a stand-alone machine or via a multi-user network were also available. As a result, the installation procedure proved to be quite flexible, installing the necessary database lock files in the appropriate locations according to the user-selected installation criteria.

Functional Enhancements

Functional enhancements involved alterations to The HWL – usually in response to survey suggestions – intended to improve program utility for the users (students and instructors). There were also changes made to the program by the development team following the system testing in order to increase the modularity of the program. Several prominent examples of functional enhancements follow.

Time Recording: After the program had been used by instructors in multiple classrooms, it was decided that instructors would benefit by being able to see how much time a student had spent in each section of The HWL – particularly the classroom section. To do this, a clock-triggered procedure was added to the software, along with five new variables: homeTime, pracTime, testTime, classTime, and idleCount. The procedure is performed once every second and, depending on which section of The HWL the user is currently in, increments the appropriate variable by one. The idleCount variable is used to ensure that excessive idle time is not included. This information is written out to the student's SDF before they close the program and can be viewed by the instructor using the Instructor Version.

Multi-Open Enhancements: Enhancements were also made to the Multi-Open option of the Instructor Version. The Multi-Open option, which allows instructors to view multiple SDFs simultaneously, was modified based on instructor recommendations. Statistical data was added on each student so that an instructor could quickly ascertain how many problems the student had started and completed as well as a breakdown of their grades. The provisions for an instructor to save the data from all of the selected SDFs out to a CSV text file, or to copy all of the data to the Windows clipboard, or to

open the data in Microsoft Excel were also made in response to user requests.

Changing of the Student Password: From early in the program development, the Instructor Version of The HWL had allowed the instructor to view a student's password and even reset it. However, students did not have the ability to change their password after the initial setup of their SDF. This was found to be impractical and was modified appropriately.

Changing the Answer Mode: Originally, the instructor could start The HWL in "answer mode" using the Instructor Version. This enabled them to see the answers for a problem using a student's SDF. That way, if a student had a question about why they had missed a problem, they could go to the instructor and ask to see what the answer was for their particular set of randomized variables.

However, this approach required that the instructor use the student's SDF to start "answer mode" via the Instructor Version. This was found to be cumbersome, as students often forgot to bring their SDF on disk. Thus, the answer mode was modified to allow the instructor to input any set of variables into any problem and see what the answer would be. This way, the instructor could check answers for students, as well as input the original variables from the textbook for verification that the program was working the problem correctly.

MultiPrint Feature: The addition of the MultiPrint feature (Figure 8) was one of the most significant enhancements made to the Student Version. It was added in order to allow students to do two things: (1) print out multiple homework problems at once, and (2) print out previously completed but unprinted homework problems. Before this was added, students had to remember to print their homework problems before entering a correct answer. If they forgot, then they would have to start the problem over with newly randomized variables in order to print the problem. This addition enabled students to print the problems out in a batch load as well as print them out after submitting the correct answer and closing the problem.

Increase in the Role of the Database: The final major enhancement to The HWL was more of an architectural change than an addition of any new feature. One of the primary requirements set forth at the beginning of development was that the program would be modular (i.e., new subject areas implemented easily). However, after beginning to switch the engineering mechanics version of The HWL (used at TTU and UT) to a statistics version (used at MHS), it became apparent that much of the module-specific data either was hard-coded or was actually part of a background image (this was the easier approach during the initial phases of development). For instance, the number of chapters in the book was hard-coded, while the titles of each of the sections were drawn as part of a background image on the section selection screen. Because of this, a highlighted and de-highlighted version of each chapter background image had to be drawn. Changing from module to module would then require that each of these images be edited to show the new titles. While the hard-coded variables could have been reset for the new text, and the backgrounds could have been modified using a graphics editor, a more efficient approach was desirable.

Instead of hard-coding module-specific data like the number of chapters and the number of sections in each chapter, this data could be stored in one or more database tables. Many changes were made to the code to implement this new database backend. However, these were one-time changes. This approach lessened the rigor of changing from one textbook to another by requiring fewer changes to the code and less editing of background image files.

Conclusions and Recommendations

The HWL has been used successfully at UT, TTU and MHS. The concept, which is intended to encourage student practice, has been statistically studied. The study results indicate

that it improves student test scores (overall) by a few percentage points, and with certain students (e.g., students with mid-level GPAs), it improves test scores significantly. Approximately 70% of the students who used the software indicated that they prefer it over the traditional homework approach.

Early in the development, the desire for modularity was important, but not as important as functionality. In the middle stage, finalizing the interface, installation, and debugging the early code took on the primary focus. In the final stages of development, the issue of full modularity became a reality.

The Monterey High School phase of deployment/testing led to a large increase in the use of the database as described above. As those modifications were being made, a more expansive goal began to take form. Ultimately, the ideal scenario would be to change from one textbook to another without having to recompile any code. If this could be accomplished, the person making the changes would need minimal knowledge of how the program works. Instead, a software utility tool could be developed to input the new module information. To achieve this goal, the problem code would need to be interpreted at runtime. This would slow the program down some, so the gain in ease of development would have to be weighed against the loss in performance.

Much important information regarding efficient processes for creating educational software were determined throughout the project – often by mistakes, but occasionally through successful preplanning. Clearly, educational software is being shown to be efficient and cost effective.[12] However, despite the success of the software as a learning tool, two significant obstacles remain in terms of widespread dissemination. First, development of learning software is often programming intensive and may require significant initial expenditures.[5,3] Secondly, the publishing industry is changing rapidly due to the advent of Internet publishing and other web-based teaching approaches. These rapid changes produce uncertainties regarding the industry's future role in print media as well as in the relationship supplementary pedagogical instruments will have with traditional textbooks. Given the merit of the approach, however, it is felt that these obstacles will be overcome – particularly by using integrated teamwork approaches between the publisher and developer. It is hoped that the description of the development process for the HWL helps the reader to make informed decisions about the design of similar learning instruments.

References

1. Bedford, A. and Fowler, W. (1999). Engineering Mechanics – Statics 2nd Edition, Addison Wesley Publishing.

2. Borland International, Inc. (1997) Object Pascal Language Guide, 100 Borland Way, Scotts Valley, CA 95067, 1997.

3. Ellis, T.J. (2003). "Engineering and Online Course: Applying the 'Secrets' of Computer Programming to Course Development," British Journal of Educational Technology, Vol. 34, No. 5, pp. 639-650.

4. Garland, K., Noyes, J. (2004). "The Effects of Mandatory and Optional Use on Students' Ratings of a Computer-based Learning Package," Journal of Educational Technology, Vol. 35, No. 3, pp. 263-273.

5. Henderson, R.C. (2005). "Helping Students Become Proficient at Solving Fundamental Engineering Problems through Practice – The Homework Laboratory," Proceedings of the ASEE Southeast Conference, The University of Tennessee, Chattanooga, TN.

6. Huband, F.L. (1997). "Looking Back," ASEE Prism, pp. 28-29.

7. Huddleston, D.H. and Walski, T. M., (2003) "Using Commercial Software to Teach Hydraulic and Hydrologic Design,"

Computers in Education Journal, v 13, n 3, pp. 43-52.

8. Katz, S., & Lesgold, A. (1993). "Collaborative problem Solving and assessment in SHERLOCK II," Proceedings of the World Conference on Artificial Intelligence in Education.

9. McConnell, S. (1993). Code Complete, Microsoft Press, Redmond, Washington.

10. Ohlsson, L. (1995). "Practice Driven Approach to Software Engineering Education," IEEE Trans. Edu., v 38, n3, pp. 291-295.

11. Thai, C.N. (2004). "Development of a Collaborative Distance Education Classroom," Computers in Education Journal, v 14, n 1, 2004, pp. 65-75.

12. Wyman, G.A. (2004). "Snowflake USD Discovers a Cost-effective Way to Deliver Technology to All Students," T.H.E. Journal, Vol. 31, No. 10, pp. 60,63.

Biographical Information

Dr. Craig Henderson is currently Associate Professor of Structural Engineering at Tennessee Tech University. He received his Ph.D. from The University of Tennessee in 1994. Dr. Henderson's areas of expertise are in seismic engineering, structural dynamics and masonry design. He has conducted research and published papers on these topics as well as on pedagogy in engineering education.

Daniel Lowhorn is an adjunct faculty member at Nashville State Community College, Cookeville Campus. Mr. Lowhorn received his Master's degree in Computer Science from Clemson University in 2004. His area of research focused on efficient caching methods for multimedia streaming data. Mr. Lowhorn also works as an independent web developer and youth minister.

William R. Tindall holds a B.S. degree in Electrical Engineering from Tennessee Technological University. He has over 10 years of experience in multimedia applications in higher education. He is currently employed by Dealer Software Associates in Franklin, TN.

Dr. David L. Larimore is currently Professor of Education at Tennessee Tech University in Cookeville, Tennessee. Dr. Larimore received his Ph.D. from The Ohio State University in 1969. In addition to being a professor, he served as Vice President for Administration and Planning at TTU for 24 years. He has had grants and contracts dealing with the use of communications satellites to deliver graduate education to remote locations and he teaches graduate level courses in educational research design, educational measurement and evaluation, and statistics for Ph.D. students.

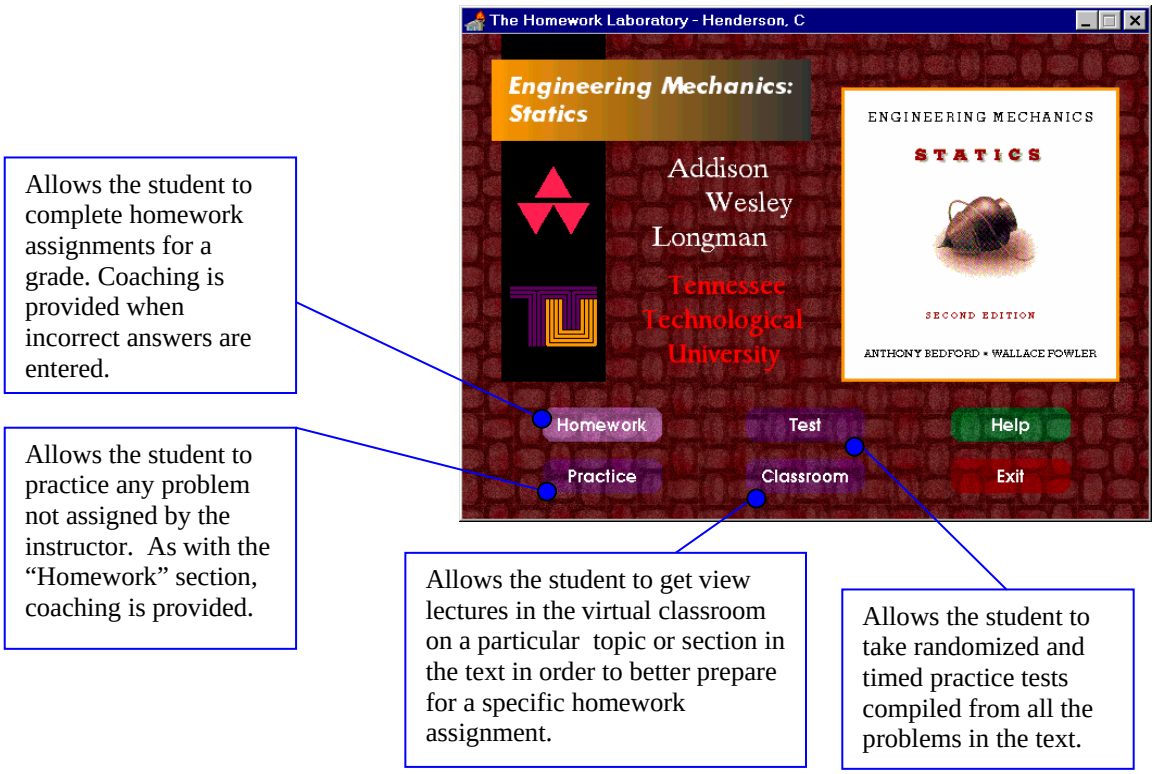

Figure 1. Student Version Main Screen.

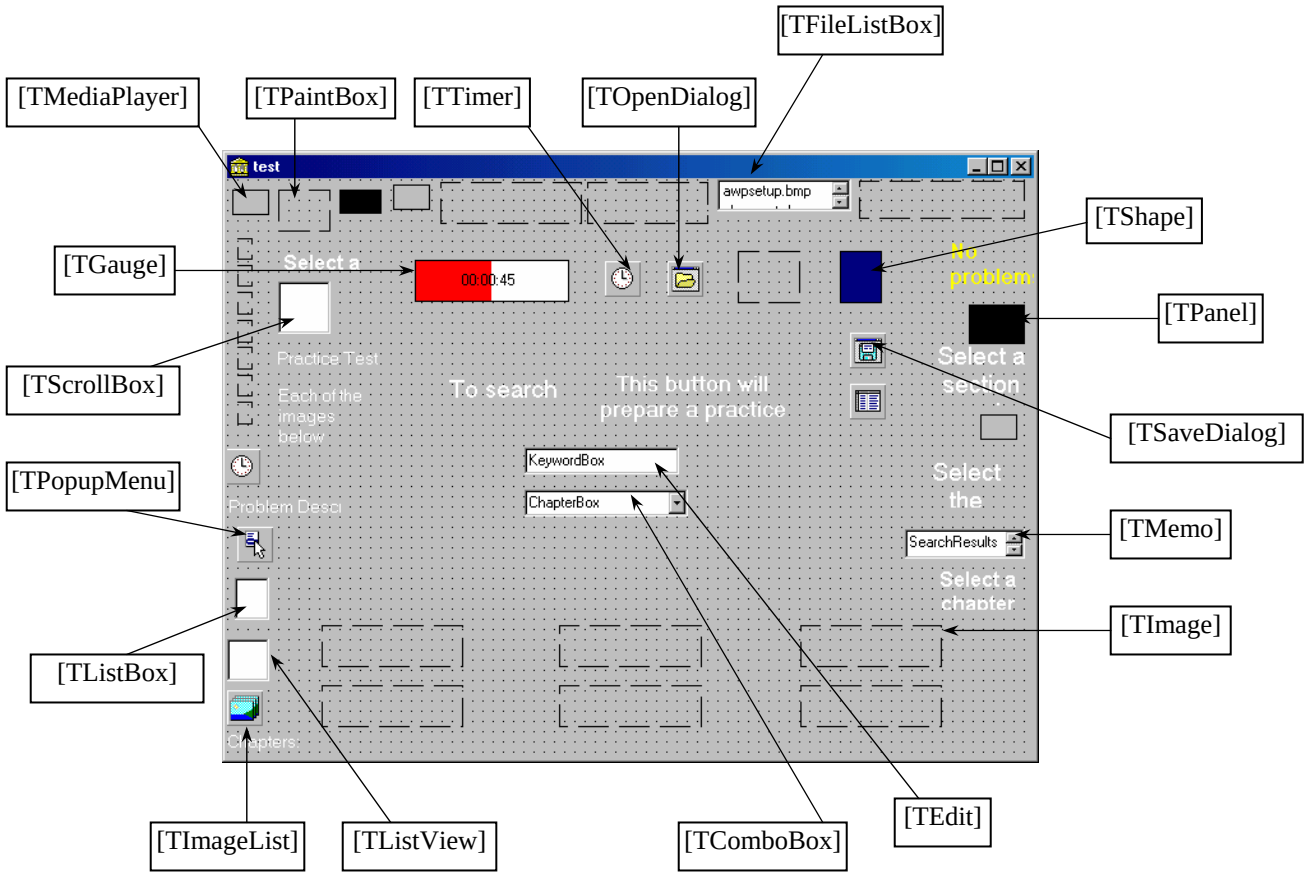

Figure 2. Main Delphi form for the HWL.

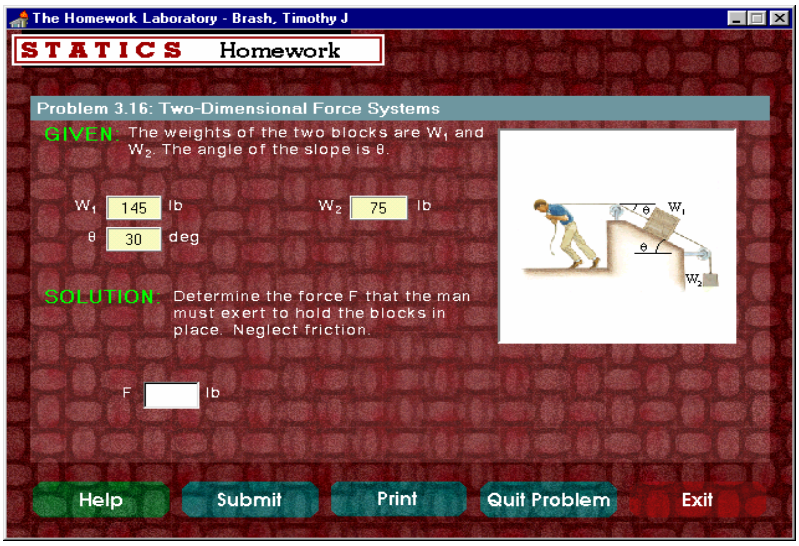

Figure 3. Typical Problem Screen.

Line

```
1   procedure problem3016;
2   var
3     W1,W2,theta:integer;
4     F:single;
5   begin
6     W1:=120+random(17)*5;
7     frmprobtemplate.PnlInput1.caption:=inttostr(W1);
8     W2:=10+random(15)*5;
9     frmprobtemplate.PnlInput2.caption:=inttostr(W2);
10    theta:=30;
11    frmprobtemplate.PnlInput3.caption:=inttostr(theta);
12
13    F:= W1*sin(pi*theta/180)+W2;
14
15    Answer[1]:=F;
16    Answerpointvalue[1]:=20;
17  end;
```

Figure 4. Code for Problem 33016.

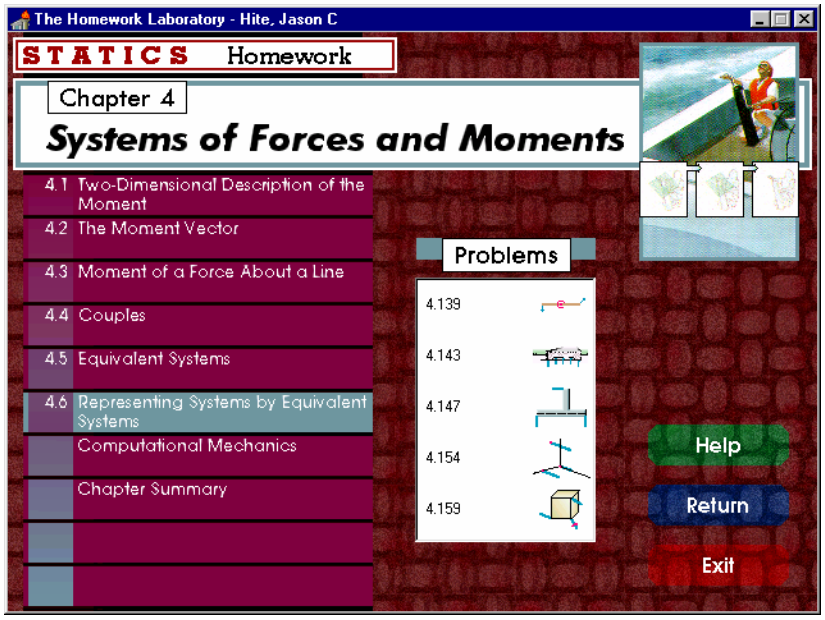

Figure 5. Problem Selection.

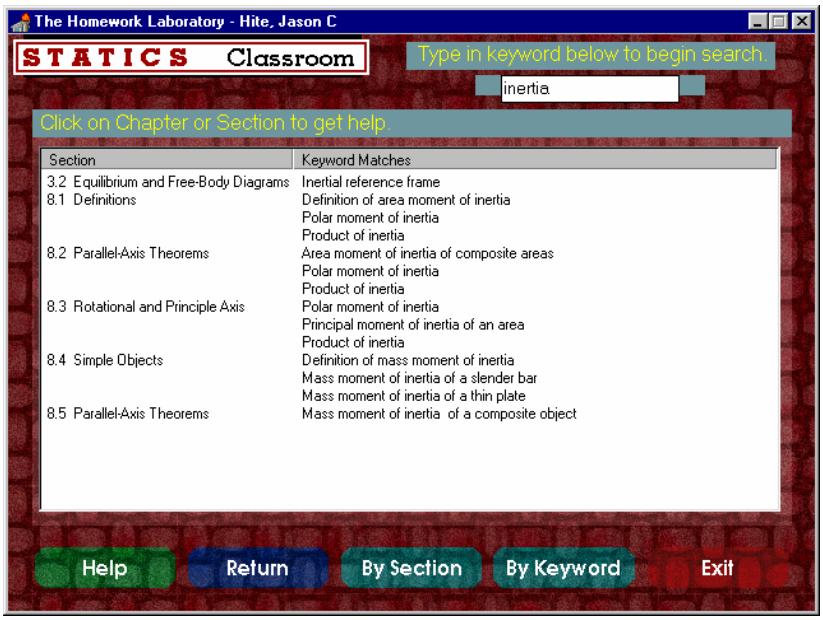

Figure 6. Classroom, By Keyword.

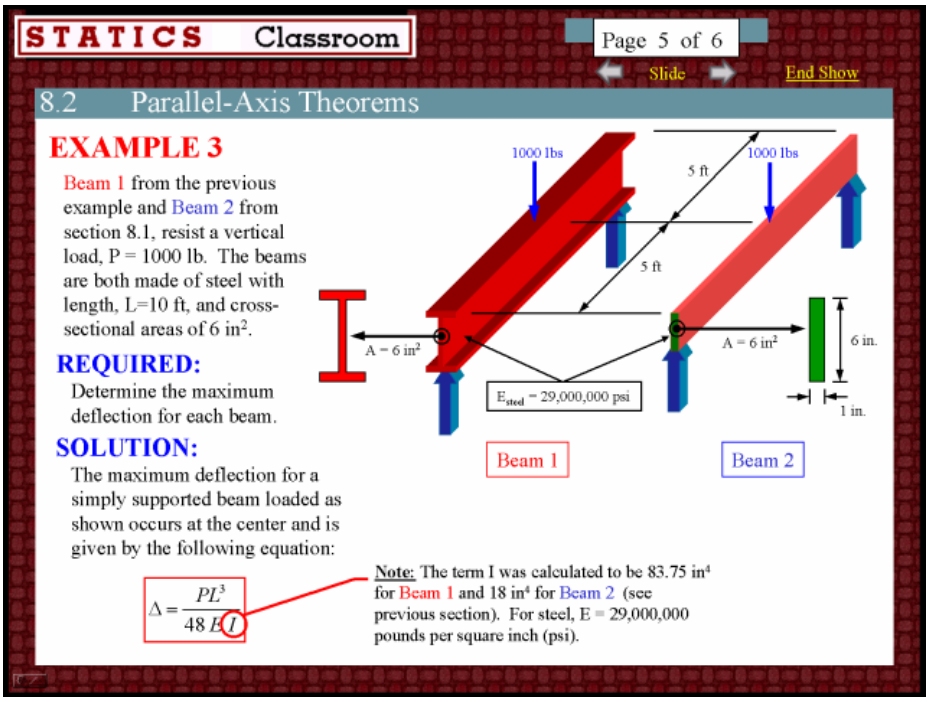

Figure 7. PowerPoint Classroom Topic.

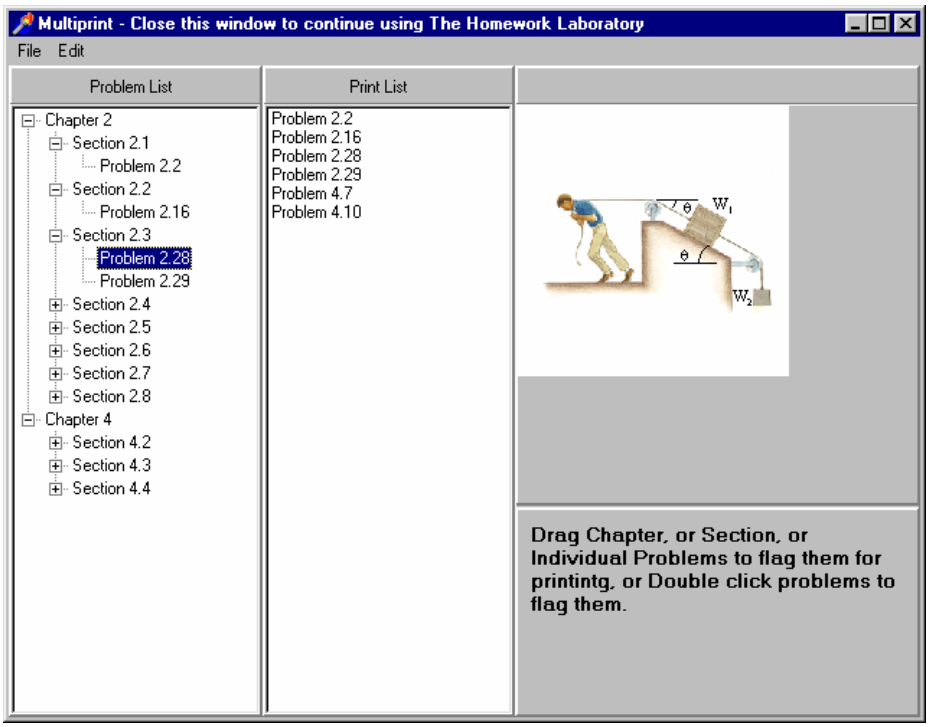

Figure 8. MultiPrint Feature.