A DYNAMIC PROTOCOL FOR COOPERATIVE COURSEWARE DEVELOPMENT

Thomas F. Thibeault
Southern Illinois University at Carbondale

INTRODUCTION

In many language laboratories throughout the country, computers have overtaken traditional audio laboratory systems as the primary instructional tool. This is likely due to the fact that computers can integrate various media within one interactive platform. Also, computers that allow students to record their voices are now commonplace and virtually every other function offered by traditional audio lab systems can be performed by computers. While this metamorphosis is taking place, powerful authoring environments designed for non-programmers are being developed. Many of these authoring tools have most of the functions of low-level programming languages, such as Pascal or C, but include enhancements, such as graphics tools and built-in drivers for multi-media peripherals. These developments have caught the attention of educators, who can now create their own CAI (computer-assisted instruction) materials to meet their specific instructional needs.

Technology has been part of the language instruction repertoire for several decades. The wide-spread proliferation of language laboratories in the 60's and 70's has provided a solid base for the implementation of CAI in the language-learning process. Since the advent of low-cost, high-performance computers and sophisticated authoring tools, a seemingly logical step in the evolution of language laboratories has taken place; language laboratories are becoming a major resource for in-house courseware development. As a result, many laboratory directors suddenly find themselves wearing the hat of CALL (computer-assisted language learning) project manager or consultant. However, the hat does not always fit without making some alterations. For example, the current generation of laboratory directors was not likely trained to wear this hat and many are not aware of

Thomas F. Thibeault, Ph.D. (Universität Salzburg) is Director of the Language Media Center at Southern Illinois University at Carbondale and an Asst. Prof. of German. *Note: The protocol is available from the author for further testing.*

formal strategies and production systems needed to take a courseware project from idea to reality. There is, therefore, a need to educate oneself about the process of courseware development. Fortunately, there is an abundance of resources[1] available in the areas of courseware development and instructional design. There is another challenge for laboratory directors serving as project managers/consultants; the process of developing CALL courseware involves many diverse areas, such as instructional design, programming, technical expertise, familiarity with several languages, research design, graphics, pedagogy, etc. Few people can claim expertise in all these areas. Most will have to rely on the expertise of others where gaps exist. As a result, the majority of high-quality courseware developed is the result of a team effort.

As soon as the team concept comes into play, the process of communication becomes priority number one. As Faiola (1989, 17) puts it, "To understand the primary deterrent to efficient courseware development, team communication must first be considered." He cites a study at Johns Hopkins University which determined that two major hindrances to effective courseware development were the time required for communication within the team and the "lack of ability, clarity and skill to convey and receive communicated information." It is safe to say that effectiveness of communication is a primary factor that determines the overall quality of the courseware production system. Poor production systems can be blamed for the poor quality of many courseware packages (Karrer, 1987, 24). There must be an effective, dynamic channel of communication between each member of the team so that goals, project status, responsibilities, assignments, deadlines, etc. are clear in the minds of all members at all times. Breakdowns in communication can lead to inefficient use of time, unnecessary repetition of work, misunderstandings, frustration and general chaos. At Southern Illinois University, we have developed a protocol that serves this need for an effective communication vehicle. It is dynamic in the sense that it can be modified to accommodate specific needs, it can be easily updated as circumstances require and updates can be distributed to other team members quickly and easily. The protocol is supported by specific strategies and authoring tools designed to reduce the development time of courseware. These will be covered in more detail later.

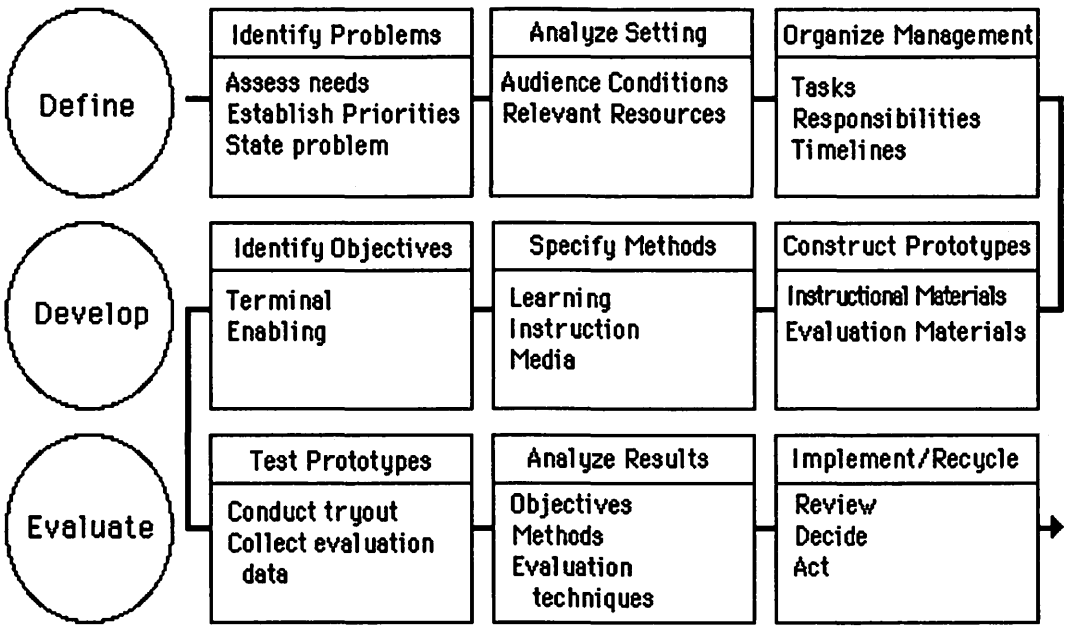
THE INSTRUCTIONAL MODEL

The protocol is based on a tri-level instructional model (Gustafson, 1991, 29) developed by the Instructional Development Institute (IDI) and shown in Fig. 1. Although the model was originally designed for developing courses, we have adapted it specifically for developing CALL courseware. The model starts with the process of defining instructional needs, analyzing the learning environment and specifying the organizational structure of the courseware. The second level focuses on development and includes establishing objectives, specifying methods of implementation, and constructing prototypes to test a given approach. The last level deals with the evaluation process which includes testing the prototype, analyzing the results, and modifying the program based on the results of the testing procedure. The diagram in Figure 1 gives an overview of the instructional model.

THE COMPUTER-BASED PROTOCOL

The courseware project protocol which we developed is actually a HyperCard computer program similar in function to IDioM (Gustafson and Reeves, 1990), a HyperCard stack which uses instructional design principles to assist in course development. Our version uses the diagram in Fig. 1 as a main menu. By clicking on the titles of each

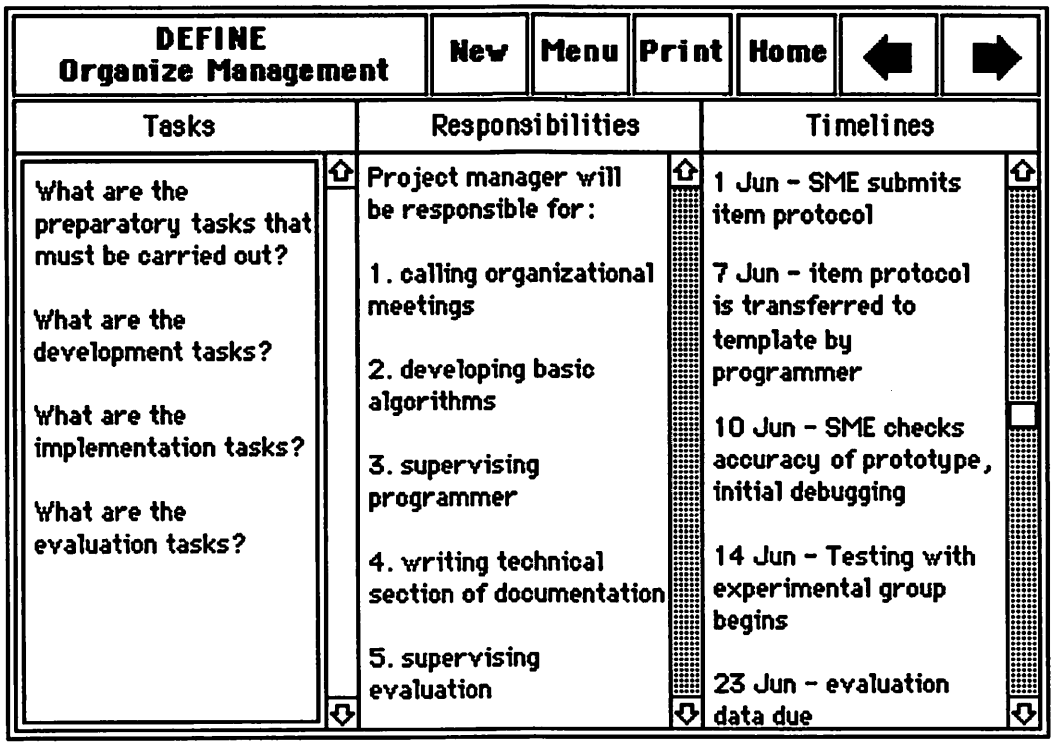
Figure 1. IDI Instructional Model (from University Consortium for Instructional Development and Technology, formerly National Special Media Institute.)



individual box, the user can branch off to a "card" that deals specifically with the selected topic. On each card there is a scrolling text field for each sub-topic where the developer can enter the appropriate information. Each field has a heading that corresponds to the sub-headings in the main menu. One feature allows users to access a list of questions and ideas under a given heading. The list provides them with queries regarding essential information that should be included under that heading. Some of these prompts are global in nature and may not apply in all cases. Others are more specific and warrant serious consideration. The prompts can be edited in response to varying instructional needs and to experience gathered over time. The idea is to maintain flexibility and let the protocol serve as a tool, without allowing it to mandate specific goals, objectives or strategies. Figure 2 below shows a sample screen dealing with organizational management on the DEFINE level. After the instructional design is complete, the entire protocol can be printed out and distributed to the members of the project team.

In addition to the project protocol, we have developed supporting materials such as templates and plug-in routines for various types of CALL exercises and activities. The templates allow developers to enter the items, directions, grammar review, etc. into special master fields. The information is then distributed throughout the template in the appropriate locations. The template contains all the algorithms needed to present the cues, process the student input and generate a performance record. Though a given template may lack flexibility in its range of applications, it can be modified to suit the specific needs of the instructor and students. Plug-in routines are programming

Fig. 2. Project Protocol - Sample Screen Showing Prompts, Features and Entries



routines that perform a specific function and can be integrated into any project. One example is a routine for analyzing student input.

For faculty members who suffer from incurable computerphobia, but do acknowledge the merits of CALL for their students, we provide paper-based protocols for listing the cues, responses, feedback, features, etc. These can then be transferred to the computer by student workers assigned to the language laboratory. Items can also be written with a word processor and then pasted into a template.

THE PROCESS BEGINS

The process of courseware development starts when an instructor approaches the laboratory director about an idea for a piece of courseware. This idea usually comes about as a result of specific instructional needs perceived by the instructor. In one case, a member of the German faculty developed an animated module that conceptualized the use of case with two-way prepositions for his bewildered students. The module contrasted a man jumping on top of a table, *Der Mann springt auf dem Tisch* (dative), as opposed to a man jumping onto a table, *Der Mann springt auf den Tisch* (accusative). This module can now be easily plugged into any CALL program dealing with this topic. Similar modules could be developed to demonstrate the concept of House Verbs in French or the difference between *ser* and *estar* in Spanish.

THE *DEFINE* LEVEL[2]

Identify Problems

During the initial brainstorming session, the laboratory director, who generally serves as the project manager, works through the project protocol with the subject matter expert (SME). The first session with the SME focuses on the DEFINE level of the protocol, though the DEVELOP and EVALUATE levels may also be discussed in general terms. During this session, it is not likely that the protocol will be completed in great detail, but working through the protocol helps clarify goals, objectives, strategies and design considerations. The major emphasis during this first session is on the target audience, defining the problem, the overall design of the program and screen design. Questions and ideas prompted by the protocol under the sub-heading *Identify the Problem* are:

a. What are the general instructional needs? This can be defined in terms of grammar concepts, cultural issues, communication skills, proficiency levels, etc.

b. Describe the target audience *(proficiency level).*

c. Prioritize the subject matter/target audience. If this is a large-scale project, which areas or audiences should be addressed first?

d. What is the nature of the problem? Are students having difficulty remembering the genders of nouns? Are their essays riddled with errors in word order? Are they particularly weak in listening comprehension? Do they have difficulty reading specialized texts, such as newspaper articles in business Spanish?

e. How serious is the problem? Does it warrant the investment in time and resources to develop software as a solution?

f. How is the target audience affected? Which learning modes are affected? What is the effect on grades and performance? What is the effect on attrition? *These items could provide useful information if included in the evaluation process.*

Analyze Setting

The next step, *Analyze Setting*, focuses on the learning environment and relevant resources:

a. Describe the learning environment. Will the students work individually, in pairs, in small groups or as a class?

b. How much time will be available per session? How will this affect motivation and concentration?

c. How many sessions will be needed to accomplish the objectives of the program? How will this affect motivation?

d. What kind of hardware do you want to use? Type of computer? How much RAM will be needed? What kind of storage devices will be required (floppy, hard disk, CD-ROM drive, videodisc player)? Will students need to record their voices?

e. Are enough computers available to accomplish the objectives within the desired time-frame?

f. What kind of supporting software will you want to use? Spelling checker? Sound editor for original recordings? Sound clips for special effects? Graphics program for creating original graphics? Animation program? Clip art?

Organize Management

The final step on the DEFINE level is *Organize Management*. The effectiveness of the courseware development process depends on clearly defined roles, tasks, assignments and deadlines. Faulty lines of communication can result in wasted effort, redundancy and lost time. Considering the great public relations effort that must often be made to sell the faculty on CALL, it is wise to keep their frustration levels down by keeping management and organization up. This section of the protocol first prompts the following questions concerning **tasks**:

a. What are the preparatory tasks that must be carried out? Writing the items, acceptable answers, feedback statements, help statements, directions, etc., and selecting the authoring tools.

b. What are the development tasks? Systems analysis, programming, developing algorithms, screen design, digitization of images and sound, etc.

c. What are the implementation tasks? Arranging for students to test the program, integrating the software into the curriculum, encouraging colleagues to adopt the software in their courses.

d. What are the evaluation tasks? Establishing the research design, keeping track of student performance, determining "bugs," content errors and typographical errors, determining the level of increased proficiency, determining the affective response of students and instructors.

The next step, according to the protocol, is to determine the make-up of the project team. The members that make up the team can vary from project to project. The core of the team is the project manager, subject matter expert and programmer. For large-scale or highly sophisticated projects, there may be a need to include an educational technologist with expertise in interfacing pedagogical theory with technological applications. A systems analyst may also be useful to help determine the best hardware and software configuration for a specific situation. Resource personnel can also be found in university media centers, computer science departments and even in business departments if the developer has entrepreneurial aspirations. In many cases, an individual may play more than one role, depending on the financial and personnel resources available; it is generally preferable to have a few good people on the team rather than a lot of partially-committed or semi-competent people. Leiblum (1989, 16) puts this in perspective when he states, "If you want a track and field team to win the high jump, you find one person that can jump seven feet, not seven people who can jump one foot."

The make-up of the team will most likely be decided by the project manager and the SME. The decisions they make collectively and individually will be determined, in large part, by the types of tasks to be carried out and the various **responsibilities** assigned to each team member. Tasks can include:

Item generation
Technical design
Screen design
Pedagogical design
Evaluation procedure
Programming
Archiving code & data
Developing time lines
Writing documentation

This list could be extended considerably, depending on the size and scope of the project.

The last major component of the organizational scheme is the **timeline**. The importance of effective time-management cannot be overemphasized. Brooks (1982,

26), father of the IBM System/360, states, "More software projects have gone awry for lack of calendar time than for all other causes combined." Virtually every grant agency requires a timeline for proposals submitted, because timelines are effective at helping researchers and developers keep up the pace and stay on task. They are a concrete reminder of accountability and commitment. Specific deadlines should be established for each major task to be accomplished. In addition to timelines, there is an abundance of project management software available through mail order catalogues and local dealers. These may be quite helpful, but something as low-tech as a hand-drawn PERT[3] chart may serve equally well for smaller projects. There are four rules of thumb that we, as developers, follow regarding time management:

Rule 1: To calculate the *actual* time needed to develop a given project, make your best time estimate for each major task listed in the protocol, sum up the total time and multiply by two.

Rule 2: The more people involved in a team, the longer the development process will take.

Rule 3: Debugging and evaluation will require more time than planning and coding together. *Planning can take twice as long as coding* (Higgins, 1984, 104).

Rule 4: The more time you spend on planning, the less time you will have to spend on debugging.

THE *DEVELOP* LEVEL

Once the preparatory tasks are defined, the development process moves to the DEVELOP level within the protocol.

Identify Objectives

The first task is to establish specific objectives for the program. The objectives should be defined in terms of what the student will be able to accomplish after working with a specific program for a certain amount of time or a certain number of times (**terminal**). The SME should also determine which preliminary skills or what background knowledge is required to accomplish the objectives of the program (**enabling**). Some programs give students the option of trying out an example or two to determine the level of comprehension. Help features, such as glossaries or concept reviews, can serve to fill in proficiency or knowledge gaps.

Specify Methods

The next step on the DEVELOP level is to specify the pedagogical approach, including methods of **learning, instruction** and **media** implementation. This can refer to a particular pedagogical theory or an eclectic collection of pedagogical principles. Hunka (1989, 14) states that "courseware should not necessarily follow a single instructional or learning theory" and that the "design of the instructional sequences should be determined by the nature of the task." According to Prabhu (1987, 108), one of the main virtues of the eclectic approach is the flexibility to adopt the best of all worlds. The empirical evaluation of the program should include an instrument for measuring the suitability of the pedagogical design. Certain questions must also be answered regarding the **instructional process**. The protocol asks questions such as:

a. How will the instruction take place? In the language laboratory with/without the assistance of the instructor? In class using an LCD projection panel? Will the introductory material be offered by the instructor in class or on the

computer as a tutorial? Will the program provide feedback after each response or only at the end of the exercise? Which learning modes (visual, auditory, tactile, kinesthetic) can be used effectively with the software.

b. Is there a particular sequence for the courseware? If so, how is the student instructed to proceed? Where does this program fit in the sequence?

c. Are students required to turn in performance records each time they use the program or only after reaching a certain level of mastery?

d. Describe your instructional strategy in detail.

There are several additional considerations to be made regarding instructional strategy. Developers should consider a variety of approaches and not cling to what Papert (1980, 32) calls the QWERTY phenomenon. This phenomenon alludes to the usual keyboard lay-out which was devised to slow down typists in order to prevent jamming the keys. With modern typewriters and word processors, the rationale for this system has long since disappeared, yet the general public has not adopted speedier and more practical keyboard lay-outs. Papert uses this analogy in the area of instructional computing when he says, "The use of drill and practice is only one example of the QWERTY phenomenon in the computer domain" (33). We now have the tools to develop materials that integrate various media and allow implementation of other CAI approaches.

Another consideration is whether the instructional strategies will be teacher-oriented or learner-oriented. Hubbard (1982, 238) describes teaching strategies as "a strategy on the part of the teacher for aiding the student in gaining proficiency in the target language." He goes on to say that a learner strategy "involves focusing more on those strategies that the learner may come to employ and control independently." Higgins (1988, 14) offers a similar perspective using his analogy of magister vs. pedagogue. When the computer functions as a magister, it parallels the teacher-oriented approach by presenting the concepts, giving examples and following a pre-defined structure. When it functions as a pedagogue, the content and flow are determined by the student, the programs are less structured and more varied in format, and the students must work at a higher cognitive level to compensate for the lack of structure.

A third consideration deals with student interaction and the computer. Although CAI lessons are generally intended for individual use, Smith (1989, 19) encourages cooperative learning experiences with computers. In this way, the interaction is not just between the student and the computer; rather the computer serves as a catalyst to stimulate interaction between two or more learners.

Overbaugh (1991, 3) specifies three types of course structures that can be applied to CAI design. These include:

a. Elaboration—the traditional method, starts with small, easy or concrete concepts and works up sequentially to large, hard or abstract concepts.

b. Inquiry Learning—the student is guided through a process to prove, modify or disprove a given hypothesis; this method is more rigid than elaboration but allows deeper processing of concepts.

c. Discovery—no pre-established hypothesis is given; students discover or rediscover principles or concepts; discovery is often the best method, but

the traditional learning environment is not conducive to this approach.

Wyatt (1982, 87) lists 14 types of CAI instructional formats that developers can implement:

1. Tutorial—introduction to new material
2. Drill and Practice—allowing mastery of material already presented
3. Game—peer competition, scoring, timing of activities
4. Holistic Practice—high level contextualized activities, such as CLOZE exercises
5. Modeling—demonstrating how to perform a language task
6. Discovery—providing situations in which linguistic generalizations can be made; e.g. inferring rules for generating comparative forms
7. Simulation—experiment with language use
8. Adventure Reading—interactive, student as detective
9. Annotation—providing a wide range of language notes on vocabulary, syntax, plot, etc. during reading or listening activity
10. Idea Processor—planning and editing outlines before writing activities or after lectures
11. Word Processor
12. On-line Thesaurus
13. Spelling Checker
14. Textual Analysis—style checkers, such as *Grammatik*

The last item requiring specification in this section of the protocol is the use of various kinds of **media** in the program. Developers must consider what kinds of media will be used (video, audio, hard-copy documentation) and whether certain kinds of media will be used to address specific instructional objectives. For example, we developed a prototype vocabulary program which uses still images and 2-second segments taken from a videodisc. The images and segments provide visual cues to prompt the student instead of using English as a meta-language. The intent was to train the student to associate the image, and not the English equivalent, with the corresponding word in the target language. Digitized sound was used to record the word on the computer so that the student could hear it pronounced by a native speaker while watching the image. Thus, the visual and auditory modes were engaged to create an association with the vocabulary item. Tactile reinforcement was added when the student typed the word out, as required by the program.

Construct Prototypes

The third step on the DEVELOP level is *Construct Prototypes*. At this stage, production of the **instructional materials** begins. A model of the program is developed where the basic algorithms, program flow and screen design are worked out. Before the actual construction, the team must decide which authoring tools to use. The SME should submit cues, acceptable correct answers, feedback statements, directions, review information and/or other help features. The protocol contains a special card for entering this information. Another card serves as a drawing board for designing the screen. The protocol should also contain a detailed narrative on how the program

should operate as well as a description of all features and functions. This information can be used to develop the printed and/or on-line documentation. The need for a prototype is not as great if the developers opt to use a tested template where many of the features and functions are pre-defined. This greatly reduces the work load and time needed to produce a finished program.

In order to determine the effectiveness of a given piece of software, **evaluation materials** or functions are included as part of the program features. A description of these features is entered into the appropriate field of the protocol. Each program includes a performance record of the student, showing the items where errors were made, the student's answers and the correct answers. Since many of our programs allow students to "peek" at the answer as a last resort, the performance record also indicates how many times they took advantage of that option. Performance records can also be printed out for further study or for the instructor's use. A cognitive evaluation of the program can be obtained using the pre-test/post-test method based on the information obtained through the performance records.

Another evaluation feature is tracking codes built into the program. These tracking codes keep track of how the students use the program, for example, which features and functions they use. This resulting data is applied to make improvements in future versions of the program.

A third measure of evaluation is a subjective questionnaire which is used to determine the student's affective response to the program. The questionnaire solicits basic demographic information, but also covers questions on how the students felt about the design of the program, the content, the instructional approach, etc.

THE *EVALUATE* LEVEL

The evaluation portion of the courseware development process is often neglected or passed over quickly. This is likely due to the logistical challenges of organizing a formal empirical evaluation. In order to have a reasonable sample size, it is often necessary to enlist the cooperation of other colleagues who may have different priorities or are overburdened with their own responsibilities. Also, many universities have policies against requiring students to participate in experiments within the context of their courses. Few students have the time or desire to take on additional work on an experimental basis. Another problem arises when determining which students should participate in the experimental group as opposed to the control group. Students in the control group may consider themselves placed at a disadvantage because they are denied access to the new materials. Students in the experimental group may feel that participation distracts them from the essentials of the course and could have an adverse effect on their grades. In spite of the many obstacles regarding evaluation, field testing and formal evaluation of the courseware are essential to determine the level of effectiveness, the response of the students and the areas for improvement. Ideally, the development of a given piece of software never really ends. The *IMPLEMENT/RECYCLE* phase should continue as long as the software is in use. This accounts for the multiple versions found in commercial software. From a pedagogical perspective, this recurring cycle is necessary to ensure that the software truly meets the instructional needs of the students.

Test Prototypes

In **conducting a tryout** of the courseware, the instructor can avoid many of the above-mentioned logistical pitfalls in the evaluation process by integrating the software into

the curriculum through the language laboratory. Elementary-level courses frequently offer an additional credit hour for laboratory work. This allows, for example, three classroom sessions and one laboratory session per week. The instructor should be present during the laboratory session to actively observe the students' performance and serve as a resource when necessary. The **collection of evaluation data** should take place in regular intervals, depending on the design of the evaluation process. This can be accomplished by requiring the students to print out their performance records. Other evaluation measures, such as tracking the students' use of specific features, can be printed out on the same protocol.

Analyze Results

The analysis of the results should be conducted in terms of **objectives, methods** and **evaluation techniques**. This can be addressed by answering the following questions:

1. Are students reaching the objectives set in the DEVELOP phase of the instructional design?

2. Are the selected methods appropriate for the stated instructional goals? How do students respond to this method? Are there other approaches worth considering?

3. Is the evaluation design appropriate for determining the effectiveness of the courseware? Do the evaluation data provide the information being sought?

Implement/Recycle

The structure of the protocol/instructional design model implies a linear approach to development, but the last block, labeled *Implement/Recycle*, indicates that the maintenance of the software is a cyclical process. In effect, the development process never ends, nor should it end if the long-term goal is to produce courseware that truly meets the instructional needs of the teacher and the learning needs of the student. The ability to constantly field test and improve a piece of instructional software within one's own academic setting is probably the best justification for in-house courseware development. As a result of this process, the quality of such software can even exceed that of commercially produced products.

CONCLUSION

Foreign language educators and language laboratory directors who wish to delve into the high-tech craft known as courseware development now have the tools to do so. The importance of using protocols as a tool to facilitate communication among project team members cannot be overemphasized. The dynamic protocol described here not only facilitates communication, but also feeds developers with essential questions and ideas for consideration in all areas of courseware development. It can have a great influence on the effectiveness of the production system and on the quality of the courseware produced through that system.

NOTES

1. ERIC lists over eighty references under "Courseware Development." These references are geared more toward the academic side of courseware development. There are also several books written for academic developers, software engineers and computer scientists. These tend to be very technical, but they contain a wealth of information worth perusing. I highly recommend the book, *The Mythical Man-Month*, by Frederick Brooks, Jr.

2. Please refer to Fig. 1 for an overview of the individual components and their corre-

sponding sub-headings in this section. The main components listed under each sub-heading are printed in bold in the narrative.

3. An organizational chart showing a sequence of tasks and corresponding deadlines.

REFERENCES

Brooks, F. P. *The Mythical Man-Month.* Reading, MA: Addison Wesley, 1982.

Faiola, T. "Improving Courseware Development Efficiency: The Effects of Authoring Systems on Team Roles and Communication." *Educational Computing* (Aug. 1989): 16-19.

Gustafson, K. L. *Survey of Instructional Development Models with an Annotated Bibliography.* 2nd ed. Syracuse: Information Resources Publications, 1991.

Gustafson, K. L. and T. C. Reeves. "IDioM: A Platform for a Course Development Expert System." *Educational Technology* (March 1990): 19-25.

Hall, K. A., R.C. Comer and J.A. Merrill. *Taxonomy of Instructional Strategies for Computer-Based Education.* Minneapolis: Control Data Corporation, 1981.

Higgins, J. *Language, Learners and Computers.* London: Longman, 1988.

Higgins, J. and T. Johns. *Computers in Language Learning.* London: Collins ELT, 1984.

Hubbard, P. L. "Language Teaching Approaches, the Evaluation of CALL Software, and Design Implications." In *Modern Media in Foreign Language Education: Theory and Implementation,* edited by William Flint Smith, 227-254. The ACTFL Foreign Language Education Series, Lincolnwood, IL: National Textbook Company, 1987.

Hunka, S. "Design Guidelines for CAI Authoring Systems." *Educational Computing* (Nov. 1989): 12-17.

Karrer, U. *"In Search for Quality Enhancement Factors: An Exploration of Production Systems for Quality Courseware Development. Report of an Informative Trip through the U.S.A. in Spring 1987.* Dissertation, U Zürich, 1987.

Leiblum, M. D. "Some Principles of Computer-Assisted Instruction, or How to Tame the Flaming Beast." *Educational Computing* (March 1984): 16-18.

Overbaugh, Richard C. "Research Based Guidelines for Computer Based Instruction Development." Boston: Eastern Educational Research Association, 13 Feb. 1991.

Papert, S. *Mind-Storms.* New York: Basic Books, 1980.

Prabhu, N. S. *Second Language Pedagogy.* Oxford: Oxford University Press, 1987.

Smith, P. E. "Some Learning and Instructional Theory Considerations for the Development of Computer Related Instructional Materials." *Educational Computing* (Nov. 1989): 18-19.

Wyatt, D. H. "Applying Pedagogical Principles to CALL Courseware Development." In *Modern Media in Foreign Language Education: Theory and Implementation,* edited by William Flint Smith, 85-98. The ACTFL Foreign Language Education Series, Lincolnwood, IL: National Textbook Company, 1987.