CAI Development: The Experience of the University of North Carolina Courseware Development Project

Frank Dominguez and Margret Hazen
The University of North Carolina

ABSTRACT: The University of North Carolina (UNC)/IBM Courseware Development Project has been successful in the creation of high-quality college level educational computer materials. This paper provides an overview of the development of the Project, of the difficulties encountered along the way, and of the solutions found to overcome them. It advocates the use of a team approach to the development of quality computer materials, and provides a model to follow.

KEYWORDS: University of North Carolina, IBM, team approach, faculty incentives, courseware development, authoring tools, educational benefits, Spanish MicroTutor

1. The Development of the Microcomputing Project at the University of North Carolina at Chapel Hill

In 1981 a small group of University of North Carolina (UNC) faculty members met informally with Dean William H. Graves to discuss the need to introduce computing across the curriculum. The group was aware of the educational potential of computer-based instructional materials, but realized that the cost of development and the lack of faculty incentives had kept the quality of available materials low and production local. Furthermore, the UNC College of Arts and Sciences did not yet have a coherent policy concerning computers. Indeed, there was not much knowledge about how or where computers could be used in the undergraduate curriculum, or of how a state institution traditionally strapped for resources, could acquire the large number of machines needed for
undergraduate instruction. One solution to both problems was to seek outside help.

At the time (1981) there were discussions of massive price reductions in the microcomputing industry and tax breaks for the educational computing market. Apple had implemented a hardware grant to educators through the Apple Foundation, and other companies were considering the desirability of a closer partnership with centers of higher education. Of the companies approached by UNC for funding, IBM was the most receptive. Negotiations on a contract between the University and IBM began in November of 1982. The resulting initial three-year contract, signed one year later in November 1983, provided money and machines to the General College of the UNC College of Arts and Sciences in exchange for a pledge to put forth "best efforts" to produce computer-based educational materials in the humanities and sciences.

The vagueness of the language of the contract reflected UNC’s and IBM’s uncertainties surrounding the venture. The University did not know whether a faculty unfamiliar with the use of computers would be willing and able to produce computer-based materials. IBM was not sure about what types of materials would be needed in a university context. Under the guidance of Dr. William H. Graves and Dr. Margret Hazen the College began the process of assembling a team of experts in programming, educational courseware design, and graphics; as well as support groups to manage computer laboratories and provide maintenance for hardware and software.

The first ten instructional software projects undertaken were in foreign languages, English, and mathematics. Another fourteen were added in January of 1984. The UNC/IBM Courseware Development Project encompasses projects in the Romance Languages Department (Spanish and French), the Classics Department (Latin), and the German Department. Small prototype modules produced by the end of the first summer (1984) proved successful enough to warrant negotiations for an extension to the IBM contract, and by May 1985 the UNC College of Arts and Sciences was beginning a second round of projects as well as holding monthly seminars aimed at dissemination of the information and expertise gained by the project to faculty and administrators from other institutions. Within the last two years, the Courseware Development Project has become a model of the development process and this model may prove useful to other institutions seeking to develop their own computer-based educational materials.
II. The University of North Carolina Project Experience: The Team Approach

"Unlike books, computers actively respond to students’ ways of learning; also unlike books, they consume resources and cry out for systematic planning."  
(Pusack & Otto, 1984, p. 195)

Over the last three years, the UNC/IBM Courseware Development Project has found that the ingredients for successful instructional software development at the university level are faculty, staff (including programmers and other support staff), software, hardware, and space. All components are necessary, and will be discussed in detail. More specifically, faculty, programmers, and staff can form a team dedicated to the production of quality courseware made possible by the integration of their various talents.

The team approach has several advantages, for example, adding a centralized professional staff to the instructional software development process brings specialized skills to the projects. Successful processes and products, once developed, can be communicated to other projects. This results in the development of increasingly higher quality software and a reduction in the time required to finish a project. Similarly, undesirable or unsuccessful processes or products can be avoided with each new project development. It is far too monumental a task to expect one individual to acquire all the necessary knowledge for instructional software development in as fast, intensive, and thorough a manner as can be acquired by a team. As Ally (1985) has indicated, multi-member teams have encouraged the development of valid, high-quality finished products, with efficiencies accrued over time.

The team approach, however, is not without its critics. The major argument put forth against it is the expense. While hardware trends lean toward greater power and speed within a smaller amount of space (Gerola and Gomory, 1984) and costs continue to decline with this miniaturization trend, the costs of software production do not (Wyatt, 1983a). Programming labor costs are increasing, software development project management skills are expensive and difficult to find, and faculty incentives to produce materials are low.

Assembling a team of professionals is a costly proposition. Salaries for specialists or professionals range from $20,000 to $30,000. Within the UNC/IBM Courseware Development Project, funding and training of such a staff became an early priority, especially in view of the decision to rely on student programmers. However, few institutions can afford to invest the funds for specialized teams, and one result has been the scarcity of high-quality instructional software.
Staff

A primary function of the staff is the recruiting, training, and monitoring of the project's student programmers. Supervised student programmers have proven to be a valuable asset to the team. They allow faculty to focus on content preparation and they produce a larger amount of well-documented programmed code than most faculty members have the time or inclination to produce. Programming has been in almost all instances delegated to the student programmers under the supervision of a full-time programmer/analyst.

Three types of programming support were available to project participants. One type involved recruiting undergraduate students in math classes and later in computer science classes for an internship program within the College of Arts and Sciences. These students were assigned to projects and worked during the semester with faculty members producing courseware. Since the internship was a credit course, the need for supervision from faculty and staff was high. The best intern programmers were offered full-time, paid summer programming positions, and this full-time experienced support formed a second type of programming support.

In addition to undergraduate programming support, some graduate students who were familiar with a project's content area and possessed programming skills were hired by the project. The programming for two of the language projects (Spanish MicroTutor and Micro-Review in French), for example, is in the hands of graduate students in Classics. Their knowledge of language teaching strategies has proven invaluable to the project, and has made communication between the faculty developer and the programmer much easier. This matching of the programmer's academic background to the project at hand, though difficult to replicate, is very desirable, and has become a third type of programming support.

Because resources like space, hardware, and software are required for instructional software development, the management of these resources becomes a staff function that takes increasingly larger amounts of time as the number of projects increases. Ordering, registering, taking inventory, and updating the hardware (microcomputers, monitors, and printers) and software (word processors, utility packages, and programming languages) for ten projects magnifies geometrically when the number of projects approaches fifty. Communication among ten projects can remain relatively informal, but communication among fifty projects takes incrementally more time and must become more formal. Project coordination and management of resources can easily become a full time job for some staff members.
Faculty

Faculty within a predominantly research-oriented institution like UNC-CH find themselves in a difficult situation. Although instructional software development is generally assumed to involve faculty as authors and often as programmers, within a university situation the incentives do not currently exist to allow them to work at an optimum pace and, if they exist, they are not commensurate with the efforts expended (Emery, 1984; Gilbert, 1985; Weible, 1983).

Faculty incentives to undertake the time-consuming and time-intensive effort of instructional software development are few. In the case of the UNC/IBM projects, compensation took several formats. Faculty members received an IBM PC or XT, word processing software, courseware development tools, and programming support for a period of three years. Additional incentives were provided in the form of:

1) Staff support or assistance with hardware and software
2) Departmental leave time
3) Extra graduate assistant or teaching assistant help
4) Software distribution assistance

Leave time was the most desirable incentive, but only external funding allowed for that. Individual faculty situations varied tremendously and ranged from no departmental leave to a two year leave for the Spanish MicroTutor project faculty member funded by the Department of Education. This grant also provided year-round full-time programming support for two years.

At the University of North Carolina the Courseware Development Project began with the assumption that faculty members would be integrally involved with the actual development of software, either at the level of authoring the software or at least at the level of being able to comprehend and add to the processes that the programmers used. The project faculty provided the control and leadership necessary to retain ownership of the project. Therefore, two features of an authoring tool were particularly important to the content expert or the faculty member: the quality of the documentation and the ease of use of the tool.

Software

As Wyatt (1983b) indicates, there are basically three approaches to development that could have been taken: 1) using authoring systems, 2) using authoring languages, or 3) using general purpose languages. The general purpose languages considered were BASIC and Pascal; the only authoring language available in 1983 was PC PILOT. The authoring systems examined
included Private Tutor, Torricelli, Trainer 3000, and Easylearn.

The languages and systems mentioned above were evaluated in terms of their usefulness and practicality for the proposed faculty projects. They were evaluated along four dimensions: 1) their usefulness to the content specialist or instructor, 2) their usefulness to the instructional designer or analyst, 3) their usefulness for the administrator of the instructional software, and 4) their usefulness to the programmer (Hazen, 1986).

From an instructional design perspective, the need for an easy-to-use authoring tool could not preclude certain other desirable and instructionally sound features such as rapid response time; the flexibility to provide variety in format or type of instructional software (e.g. drill, practice, simulation, tutorial, problem solving); the flexibility to provide structural features such as help and hint routines, skip backward (review) and forward routines, restart routines, glossary or dictionary routines; the flexibility to provide screen windows or viewports to allow instruction to evolve on the screen; and the flexibility to provide a variety of input and output formats (mouse, alternate fonts, colors or highlighting features on the screen).

Programmers need a language that provides straight-forward implementations for the necessary instructional features. They require sophisticated and easily implemented answer analysis capabilities, sophisticated and easily implemented text or screen presentation methods, rapid "revise, test, and revise" cycles, randomizing functions, and interactive modes. Many complex simulations and problem-solving instructional software formats require sophisticated mathematical functions, input/output routines, and data structures. In addition, when training time is an issue, prior familiarity with a language may become a factor; and maintenance is easier with some tools than others. For long term administrative needs, such features as student record keeping or management capabilities to collect and analyze student data; as well as technical help, cost, and transportability of the language are also factors.

Initially two languages were chosen, PILOT and BASIC. A third language, Turbo Pascal, emerged several months later, and in many instances replaced BASIC as a language for faculty projects. Using both PILOT and Pascal became the best way to provide the features that individual projects required.

Special purpose authoring systems have emerged for the foreign language content area and more continue to be written (Lines & Martine, 1983; Mydlarski & Paramskas, 1984; Holmes, 1983). However, PILOT still offered greater possibilities for the foreign language projects. The PILOT language allows a
programmer to create a rough working module of a rather complex lesson in about a day. Revision and debugging, of course, take much longer. PILOT offers the ability to construct and organize screens using both color and text, thereby making the presentation of materials easier. In addition, PILOT is more easily understood by the layman, and it is designed to be changed fairly easily.

**Hardware and Space**

At a campus without previous computing experience in the humanities, the first computing decisions to be made usually involve hardware. In the case of the University of North Carolina, both hardware and software choices for instructional software development were influenced by the need for alternate character sets for Romance languages. The IBM PC produces these symbols without generating a graphic character set. Similarly, a word processor was selected that could take advantage of the foreign language alternate character symbols.

Space is not readily available at the University of North Carolina at Chapel Hill (one of three resources in high demand) and therefore, locating and acquiring laboratory space for testing the software, as well as space for programmers to work in, has been a time-consuming and difficult task. Additionally, the Chapel Hill campus is relatively old, and therefore each microcomputer laboratory needed to be rewired and redesigned with respect to air flow.

**III. Conclusion**

Benefits, both anticipated and unanticipated, have accrued from faculty involvement in courseware development. First, there is the renewal of spirit that comes from entering a new field. Concomitant benefits are the opportunities to present at new conferences and to meet a different set of colleagues, generally across disciplinary lines. Over time, faculty members also noticed that their teaching efforts were improving as an indirect consequence of their participation in the project. Weible similarly indicates: "teachers may find their classroom presentations becoming more focused and effective" (Weible, 1983, P. 64).

Another skill that evolved over the project’s first years was that of instructional design. Faculty had little information on how to best use this new instructional medium. Questions such as how much text to put on the screen, what colors to use, what sound if any, how much control to be placed in the user’s hands, what support to provide [dictionary, thesaurus, help screens, indexes, etc.], and how to evaluate results, require non-traditional teaching strategies and the knowledge of a courseware designer. Such a person can
communicate and encourage the application of significant design principles to produce quality teaching materials for computers (Hazen, 1985).

Specialized skills to communicate and enforce significant design principles that lead to higher quality software have emerged throughout the project. One person such as a faculty member, cannot acquire the necessary knowledge in as fast and intensive a manner as a team. A team allows for valid, high-quality finished products, with efficiencies accrued over time. The valuable skills built up become increasingly transferable to new team members because of the documentation developed simultaneously.

While authoring systems remain highly desirable from the productivity and cost perspective, they are not yet flexible enough to produce high-quality educational software that responds to particular situations. In our view, the best approach is to provide a professional Support team for the content specialist. The UNC/IBM Courseware Development Project approach has succeeded in producing high-quality instructional software even though it began in the context of relatively little faculty experience with instructional courseware production.

Notes

1 A common approach has been to develop brief modules—ones that can ideally address research needs as well as teaching purposes. If they are distributed, it is among peers. Few large scale commercial projects exist. Major publishers in the language area, for example, have opted to produce simple to program drill exercises rather than expensive tutorials. High-quality software considerable quantity remains difficult to find; "...the currently available foreign language microcomputer courseware is meager in quantity and generally unimpressive in quality." (Weible, 1983, p. 63).

2 Spanish MicroTutor is an interactive tutorial to be published by Harcourt Brace Jovanovich that covers the essential points of first-semester Spanish grammar. Micro-Review in French is an interactive tutorial to be published by Harcourt Brace Jovanovich for the beginning student of French. Glossa is a system that takes text in any language and formats associated files to aid the reading and translation of foreign language prose to poetry. Modules are available in Latin and German.

3 Very often, today’s authoring packages sacrifice power for ease of use (or vice versa). Occasionally, a system is introduced which compromises both in order to offer the optimum facility. This is rare. Too many systems, as evidenced by the resulting courseware, are frame oriented, text based, with a fixed educational strategy. In addition, they often use unsophisticated answer diagnosis and lack student control features. Even when excellent tools are
provided, there is another aspect that leads to the unsatisfactory state of courseware today. Authors who learn the detailed procedures of a specific tool "have too often neglected to learn about instructional design, man-machine dialogue disciplines, screen design and evaluation techniques." (Heaford, 1985, pp. 4-5).

References


Authors' Biodata

Frank Dominguez is Associate Professor of Romance Languages at the University of North Carolina at Chapel Hill. He is the author of *Spanish*.
MicroTutor: A First Year Interactive Spanish Grammar Tutorial, soon to be published by Harcourt Brace Jovanovich.

Margret Hazen is the Director of Instructional Computing in the Microcomputing Support Center of the University of North Carolina. She has had over 15 years of computing experience and over 10 years of instructional computing experience and is the author of several articles concerning computer-based instruction design and development.

Authors' Addresses

Dr. Frank Dominguez
Department of Romance Languages
Dey Hall 014A
The University of North Carolina
Chapel Hill, NC 27514

Dr. Margret Hazen
Director of Instructional Computing
Microcomputing Support Center
CB# 3450, 402 Hanes Hall
The University of North Carolina
Chapel Hill, NC 27514